32 BASIC Programs for the Exidy Sorcerer

Tom Rugg

Phil Feldman

Kevin McCabe

32 BASIC Programs for the Exidy Sorcerer

Tom Rugg, Phil Feldman
and Kevin McCabe

dilithium Press
Beaverton, Oregon

To
Marie South Williams
and
Betty Lou McCabe,
who pushed, prodded, and poked for perfection—
but were satisfied with the results anyway.

10 9 8 7 6 5 4 3 2 1

ISBN: 0-918398-35-5
Library of Congress catalog card number: 80-68532

Printed in the United States of America.

dilithium Press
P.O. Box 606
Beaverton, Oregon 97075

Acknowledgements

My thanks to:

– Bill Rose, Jim Rose,and Andy Papas of the LaGrange (Illinois) Byte Shop, for their technical assistance, encouragement, and unending good humor throughout this project, and

– Margo Dunton, for producing each of the photographs in this book despite my fumble-fingered assistance and advice.

C. Kevin McCabe

Our thanks to:

– Our wives and families, and

– John Craig, Asenatha McCauley, Herb Furth, Freddie, and Wayne Green,

for all their help and encouragement.

Tom Rugg and Phil Feldman

AN IMPORTANT NOTE

The publisher and authors have made every effort to assure that the computer programs and programming information in this publication are accurate and complete. However, this publication is prepared for general readership, and neither the publisher nor the authors have any knowledge about or ability to control any third party's use of the programs and programming information. There is no warranty or representation by either the publisher or the authors that the programs or programming information in this book will enable the reader or user to achieve any particular result.

Preface

Sitting across the room is perhaps the most expensive piece of electronic wizardry you've ever purchased. It's in a handsome cabinet, and looks quite impressive—but what does it do?

Finfrock's First Postulate of home computers covers this situation. The response to any proud announcement of purchase of a home computer will invariably be, "But what do you do with it?". If you collect ten dollars each time you hear this question, you'll be able to pay off the loan for the computer in no time.

Your answer, of course, depends on your own situation. Perhaps the computer was purchased for mathematical work, or to assist in engineering calculations and business record-keeping—those are certainly solid, conventional reasons for a computer. But maybe you want the computer to simply play games, or to help with your family's education. Or maybe you just love having the latest electronic gadgets. Relax! Those are good reasons too, and they point out the true underlying advantage of computer systems.

A computer, by nature, is an extremely stupid beast, incapable of doing much more than consume electricity when left to itself. There are many "smarter" machines available—electronic bookkeepers for banks, automated testing and processing equipment in factories, and many others. Each of these other machines is a genius in its field, but *only* in that field. On the other hand, a computer can be "educated" in *any* field, and it can change fields of expertise as fast as new instructions can be given to it. In other words, a computer is a general-purpose problem-solver, and is not restricted to any particular use. So, even if you purchased the machine for business use, it can still be a game-playing or teaching machine after hours.

The Exidy Sorcerer was designed with just such versatility in mind. The central processing unit that is the "brain" of the Sorcerer is a Z-80 microprocessor, a computer-on-a-chip unimaginable twenty or even five years ago. Its abilities include more than enough processing ability for business use, and sufficient speed for even extremely complex scientific and statistical uses. For "fun" uses, the Sorcerer also contains a sophisticated control system that can customize a television screen into an artistic work or a game board. The professional-style keyboard facilitates communications between the user and the Sorcerer for all types of uses by allowing most keys to represent up to four different characters.

This all sounds great, but there *is* a catch to it. To use all of these fancy abilities, to teach the machine to become an expert in a selected field, requires one key ingredient—a computer program. A program is a series of instructions to the machine. These instructions tell the machine how to combine its built-in general-purpose abilities into a meaningful series of steps which, when performed in the proper sequence, will play a game, balance a checkbook, or mimic the specialized function of a tachistoscopic reading machine.

Where do programs come from? Programmers, of course (Finfrock's Third Paradox is "Which came first, the program or the programmer?"). There are several ways to get programs:

1. Hire a computer programmer—if you can afford it! Good programmers are expensive and hard to find. The cost of three or four moderately complex programs could easily exceed the cost of the Sorcerer itself.
2. Learn how to program. This is the best method, but it takes some time. There are lots of programming books available, with some of the best being published by dilithium Press for those readers willing to learn on their own. If a more traditional approach is desired, most colleges and many high schools offer night courses at low cost.
3. Buy "canned," prewritten programs for the uses you desire. This is far cheaper than hiring a programmer for custom work, because all buyers share the cost of developing the program. However, the price still isn't cheap enough to

ignore, particularly if several dozen programs are desired. Another problem is uneven quality—there's no guarantee that a hundred-dollar program will perform any better than a ten-dollar one. Canned programs are often difficult to modify for any particularized needs, making it hard to find programs that do *exactly* what is desired. Even if you find good programs written in the BASIC programming language, there's still no assurance that they will run properly on the Sorcerer. As you'll soon learn, BASIC is a "standard" language only to a degree, with different computers using slightly different dialects of the language.

What's the solution? Hopefully, this book! All three of the alternatives noted above have been combined, to produce 32 interesting and useful programs touching on a variety of fields. The concepts for these programs were developed by two computer professionals—Tom and Phil—then adapted to the Sorcerer by Kevin, a long-time computer "nut" and owner of one of the first Sorcerer machines.

Just as important, these programs have more than the bare "listings" or "print-outs" of the necessary instructions. Each program listing is accompanied by much more information than normally found with canned programs. Even without programming skills, this information will enable you to customize the programs, and to learn how they operate. Even more extensive modifications can be made, following the suggestions with each program, after you learn a little of the BASIC language. No matter whether you are a complete beginner or an old hand with computers, you'll find plenty of "goodies" to hold your interest.

But that's enough of the sales pitch. Our main point is that the computer is an incredibly flexible machine, and shouldn't be considered to be tied to any particular use. This book presents a wide range of possibilities, yet it barely scratches the surface.

Open you eyes and your mind, and let the Sorcerer weave its spell. Play mental games against the computer (WARI, JOT). Evaluate your financial position (CBOOK, LOAN), and select a course of action (PICKS). Expand your vocabulary, or improve your reading speed (VOCAB, TACHI). Check the health of your car (MILES) or your emotions (BIORH). Convert the machine into an arcade game (WALLS, RACER) or into a work of art

(KLEID, SPARK). Have fun with an animated cartoon (ACROB), or get down to serious business with statistical analysis (STATS).

In short, enjoy! But please be careful—you may soon be spellbound!

How to Use This Book

Each chapter of this book presents a complete computer program designed to run on any Exidy Sorcerer home computer with at least 16K of memory. There are eight sections in each chapter:

1. **Purpose**: Explains what the program does, and how it might be of use to you.
2. **How To Use It**: Gives the details of how to run the program. Explains the various options which can be selected by the user. Details any limitations of the program, and its suitability to particular uses.
3. **Sample Run**: Illustrates the results of the program when used in a typical manner.
4. **Program Listing**: Provides a "listing" (or "print-out") of the program's BASIC language commands. These instructions must be stored on tape for later runs.
5. **Easy Changes**: Details simple changes to make the program work differently. Even without programming skills, these changes can be made easily and quickly.
6. **Main Routines**: Explains the general logic of the program, to allow a user to understand how the program operates. Gives the line numbers and brief explanations for all major portions of the program.
7. **Main Variables**: Explains the values associated with each of the major variables in the program.
8. **Suggested Projects**: Provides one or more ideas for major changes to the program. Some understanding of the BASIC language will be required to make these changes.

To use any of these programs on a Sorcerer computer, only the first four sections of the chapter will be needed. The latter four provide enough supplementary information for either novice or experienced "tinkerers."

RECOMMENDED PROCEDURES

Before trying any of the programs in this book, get acquainted with the Sorcerer and the books that came with it. Learn the basics of communicating with it through the keyboard and cassette tapes. Study how to enter a program, correct mistakes, and run the finished product. Even if you've never seen a computer before, Exidy's excellent manuals will teach you the basics in just a few hours. After that, the fun begins:

1. Pick a chapter and read the "Purpose" section. If it sounds interesting, fine—but if not, there's 31 others from which to choose. If you are a newcomer to computers, try one of the short "Miscellaneous Programs" first.
2. Read the "How To Use It" and "Sample Run" sections of the chapter for more details on what the program does.
3. Enter the NEW command, or hit RESET to eliminate any existing programs already in the Sorcerer's memory. *Carefully* enter each line of instructions shown in the "Program Listing." Be particularly careful about the punctuation characters such as colons, semicolons, and commas.
4. Use the LIST command to check the program lines that have been entered. Watch for typographical errors, omitted lines, and other mistakes. Don't confuse colons with semicolons, or take an alphabetic I or O for a numeric 1 or 0 (take a moment to note the differences between these characters in the "Program Listing" sections).
5. Before trying to RUN the program, use the CSAVE command to temporarily save it on a cassette tape. Even for experienced programmers, it is unusual to enter a printed program perfectly on the first try; any error might cause the Sorcerer to go slightly crazy and "forget" all the commands you have so laboriously entered.
6. RUN the program. Does it give the same results as those shown in the "Sample Run" section? If so, congratulations—you've done a fine job, and can go on to the next step. But if

not, don't despair—read "What To Do When Nothing Works" below.

7. Once the program is running properly, CSAVE it permanently on a cassette, using the same name as the title of the chapter. The next time you want to use the program, you won't have to type it in; the CLOAD command will read the program from the tape, saving a great deal of time.
8. Read the "Easy Changes" section, and try any that sound interesting. Use CSAVE to save the new versions as well, using any name you select.
9. Use the remaining three sections for more extensive changes. If you are a newcomer to BASIC, you might decide to wait until later for this step, or ask a more experienced programmer for help.
10. In any case, enjoy yourself! If one of the programs proves particularly interesting, explore the other programs in that section as well. And don't forget—just because your *primary* purpose for using the computer is either business or mathematics, there's no reason not to enjoy the games and educational programs as well.

WHAT TO DO WHEN NOTHING WORKS

Perhaps the most frustrating feature of computers is their demand for accurate and exact instructions. A dropped semicolon in a high school composition might mean the loss of a point or two; a dropped semicolon in a computer program may have far more serious consequences.

A really serious error in a program will be "fatal"—to the program, that is, not to the programmer! A fatal error will confuse the computer so much that it will be unable to run the program. When the machine encounters such an error while the program is running, it will halt and print a message to the user, along with the number of the line with the error.

The most common error is one of syntax—that is, some type of character or usage that the computer didn't expect to find in the line. LIST the line, and check it closely. Are all of the words there, and are they spelled correctly? Have you used an I or O instead of a 1 or 0? Was a colon substituted for a semicolon? Correct any errors that you find, then RUN the program again.

Another common error is caused by the insufficient allocation of memory space for numeric or string character variables. Check the size of any arrays. If a CLEAR instruction is used near the start of the program, try increasing the number which follows it to increase the amount of memory reserved for the string.

Keep correcting any errors found, until RUN causes no more error messages. Try the program with the data from the "Sample Run" section. Are the same results produced? If so, you've done it – the program is now finished. But if not, one of two situations may exist. First, perhaps it's not *supposed* to have exactly the same results. Some of the programs, particularly the games and graphics routines, are designed to do unpredictable things to avoid repetition. The second possibility is that the program contains a "nonfatal" error, insufficient to cause terminal confusion in the Sorcerer but enough to make the results incorrect. These are a bit tricky to find, but you can normally narrow the area of inquiry by noting the point at which the error occurs. Is the first thing displayed correct? If so, the error is probably after the PRINT command for that line. Look for the types of mistakes mentioned before, particularly the sneaky typographical errors.

Now, if all else fails – read the instructions (Finfrock's Axiom). Perhaps you've misunderstood what the program is supposed to do. If not, try all the steps mentioned one more time, just in case. If that fails too, it's time to call in the experts – usually found hanging around the local computer store, and available for the cost of a six-pack or two. There's nothing wrong about asking for help, although it might be a bit rough on the ego to have some 18-year-old (who has been programming since kindergarten) point out a typo or a missed line. That's all part of working and living with computers.

A NOTE ON THE "SAMPLE RUN" SECTIONS

The "Sample Run" sections are the results of actually running and using the program, with the data values shown. Some of the runs are illustrated with photographs of the video screen; others are reproductions via the typewriter. Please note that, because of space considerations, some of these latter samples have been slightly edited, but still present the full "flavor" of the program.

A NOTE ON THE PROGRAM LISTINGS

A line on the Sorcerer's screen is 64 characters wide. For best reproduction in this book, however, lines have been limited to about 52 characters in the "Program Listing" sections. As a result, some of the lines may look a little crowded in the book.

When you enter the program lines into your Sorcerer, feel free to insert extra spaces in the lines to improve readability. The only place where spaces shouldn't be added is within the quotation marks of a PRINT or INPUT statement. Added spaces there would cause changes in the output to the screen which, in turn, might mess up the results. Otherwise, add as many spaces as you desire, up to the screen limit of 64 characters per program line.

A NOTE ON PROGRAMMING PHILOSOPHY

A curious characteristic of computer programming is that there is rarely a single "right" way to perform a given task. Most operations, such as sorting a list of values or outputting results, can be done in a wide variety of ways.

The programs in this book are the result of a cooperative effort by three authors. As a result, each programmer's particular "style" sometimes peeks through with a favorite method or technique. Beyond that, one of the points of this book is to illustrate the many ways to approach a programming problem. After gaining some experience with the BASIC language, explore some of these methods, and adapt them to your own programs.

In the meantime, further reading is highly recommended. The bibliography lists a number of useful and informative texts with more information on programming methods and languages. Check out the book section at your local computer store too. By all means, don't miss the opportunity to talk to other programmers at the computer store, or at meetings of local computer clubs. Even if you decide not to learn to program, learning some of the technical jargon and buzz words will make it easier to select and use other canned programs in the future. Who knows? You might even change your mind and decide to learn BASIC—and then FORTRAN, COBOL, PASCAL, ALGOL, PL/1, C, . . .

Contents

Section 3—GAME PROGRAMS

Match wits with the computer or a friend.

Section 4—GRAPHICS DISPLAY PROGRAMS

Dazzling visual diversions.

Section 5—MATHEMATICS PROGRAMS

For math, engineering, and statistical uses.

Section 6—MISCELLANEOUS PROGRAMS

Short programs that do interesting things.

Section 1
Applications Programs

INTRODUCTION TO APPLICATIONS PROGRAMS

Good practical applications are certainly a prime use of personal computers. There is a myriad of ways the Sorcerer can help us to do useful work. Here are six programs for use around the home or business.

Financial considerations are always important. LOAN will calculate interest, payment schedules etc. for mortgages, car loans, or any such business loan. Do you ever have trouble balancing your checkbook(s)? CBOOK will enable you to reconcile your monthly statements and help you find the cause of any errors.

Fuel usage is a constant concern for those of us who drive. MILES will determine and keep track of a motor vehicle's general operating efficiency.

By no means is the Sorcerer restricted to numerical-type applications. We are often faced with difficult decisions. PICKS forms the Sorcerer into a trusty advisor. Help will be at hand for any decision involving the selection of one alternative from several choices.

Before anything else, you might want to consult BIORH each day. Some major airlines and industries are placing credence in biorhythm theory. If you agree, or "just in case," simply turn on your Sorcerer and load this program.

QEXAM analyzes the answers to multiple-choice questionnaires and exams, question by question. Use it for test results, customer surveys, or opinion sampling.

BIORHYTHM

PURPOSE

Did you ever have one of those days when nothing seemed to go right? All of us seem to have days when we are clumsy, feel depressed, or just cannot seem to force ourselves to concentrate as well as usual. Sometimes we know why this occurs. It may result from the onset of a cold or because of an argument with a relative. Sometimes, however, we find no such reason. Why can't we perform up to par on some of those days when nothing is known to be wrong?

Biorhythm theory says that all of us have cycles, beginning with the moment of birth, that influence our physical, emotional, and intellectual states. We will not go into a lot of detail about how biorhythm theory was developed (your local library probably has some books about this if you want to find out more), but we will summarize how it supposedly affects you.

The physical cycle is 23 days long. For the first 11½ days, you are in the positive half of the cycle. This means you should have a feeling of physical well-being, strength, and endurance. During the second 1 1½ days, you are in the negative half of the cycle. This results in less endurance and a tendency toward a general feeling of fatigue.

The emotional cycle lasts for 28 days. During the positive half (the first 14 days), you should feel more cheerful, optimistic, and cooperative. During the negative half, you will tend to be more moody, pessimistic, and irritable.

The third cycle is the intellectual cycle, which lasts for 33 days. The first half is a period in which you should have greater success in learning new material and pursuing creative, intellectual activities. During the second half, you are supposedly better off reviewing old material rather than attempting to learn difficult new concepts.

The ups and downs of these cycles are relative to each individual. For example, if you are a very self-controlled, unemotional person to begin with, your emotional highs and lows may not be very noticeable. Similarly, your physical and intellectual fluctuations depend upon your physical condition and intellectual capacity.

The day that any of these three cycles changes from the plus side to the minus side (or vice versa) is called a "critical day." Biorhythm theory says that you are more accident-prone on critical days in your physical or emotional cycles. Critical days in the intellectual cycle aren't considered as dangerous, but if they coincide with a critical day in one of the other cycles, the potential problems can increase. As you might expect, a triple critical day is one on which you are recommended to be especially careful.

Please note that there is quite a bit of controversy about biorhythms. Most scientists feel that there is not nearly enough evidence to conclude that biorhythms can tell you anything meaningful. Others believe that biorhythm cycles exist, but that they are not as simple and inflexible as the 23-, 28-, and 33-day cycles mentioned here.

Whether biorhythms are good, bad, true, false, or anything else is not our concern here. We are just presenting the idea to you as an interesting theory that you can investigate with the help of your Sorcerer computer.

HOW TO USE IT

The program first asks for the birth date of the person whose cycles are to be charted. The numeric month and day are entered, along with the year. For any year between 1900 and 1999, only the last two digits need be entered.

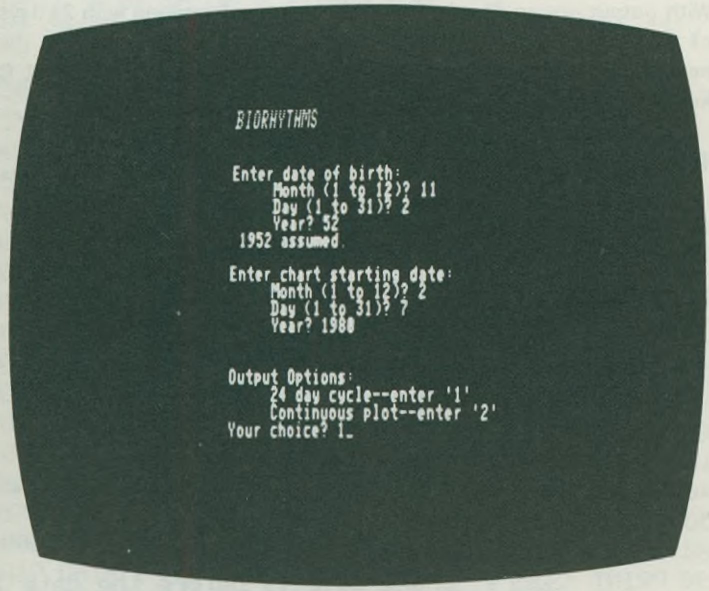
Next the program asks for the starting date of the biorhythm chart. Enter it in the same manner as the birth date. Make sure, of course, that the start date is later than the date of birth. The final input selects either a continuous plot, or a 24-day cycle.

The program clears the screen and begins plotting the chart, one day at a time. The left side of the screen displays the date, while the chart occupies the right half. The chart itself is divided into "low" (negative) and "high" (positive) areas, with critical days occurring when the plot crosses from one side to the other.

Each of the three cycles is plotted with an identifying letter. Where the curves cross, an asterisk is displayed instead of identifying letters. In the 24-day mode, 24 lines of the chart are output at a time before the program pauses. Entering **C** will cause the screen to clear and the next 24 days will be displayed. Entering **R** ends the current chart and restarts the program. In the continuous output mode, the plot fills the screen, then scrolls for each additional line until **CTRL C** is hit.

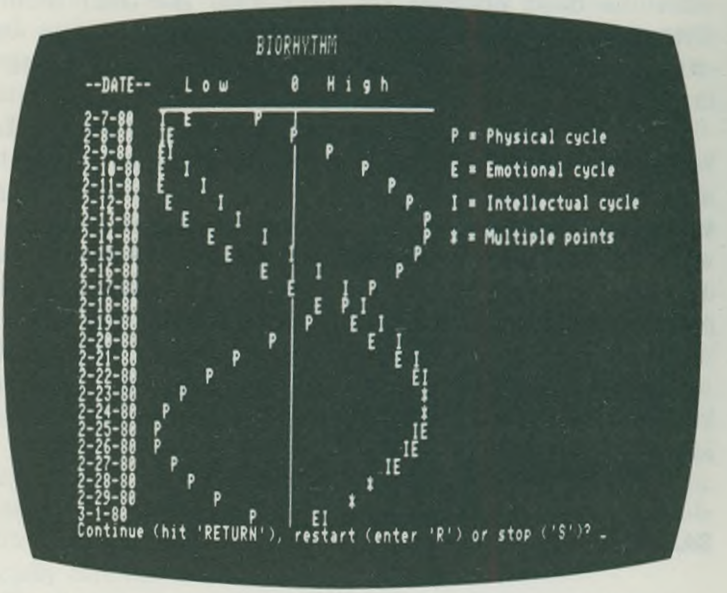
The program will allow you to enter dates within several hundred years of the present. We make no guarantees about any extreme dates.

SAMPLE RUN



```
 BIORHYTHMS

 Enter date of birth:
       Month (1 to 12)? 11
       Day (1 to 31)? 2
       Year? 52
  1952 assumed.

Enter chart starting date:
      Month (1 to 12)? 2
      Day (1 to 31)? 7
      Year? 1980


Output Options:
      24 day cycle--enter '1'
      Continuous plot--enter '2'
Your choice? 1_
```

The user's birthdate and the starting date of the chart are entered in numeric form. An output plot which runs continuously or halts after 24 lines (days) is selected by the user.



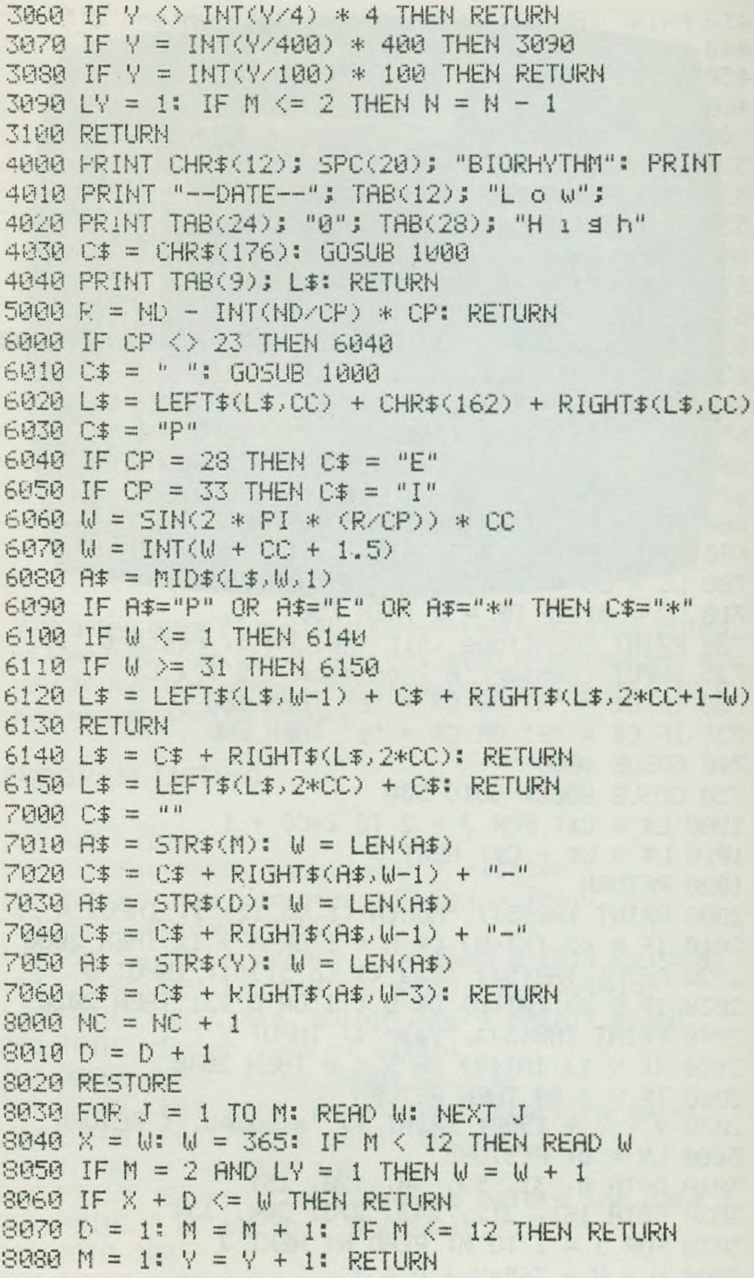
```
                    BIORHYTHM

--DATE--    L o w       0  H i g h
         -------------------------------
2-7-80    I  E      P   |
2-8-80    IE            P                   P = Physical cycle
2-9-80    EI            |   P
2-10-80   E  I          |      P            E = Emotional cycle
2-11-80   E   I         |         P
2-12-80    E    I       |           P       I = Intellectual cycle
2-13-80     E     I     |             P
2-14-80        E     I  |             P     * = Multiple points
2-15-80          E      I            P
2-16-80              E  | I        P
2-17-80                 E    I  P
2-18-80                 | E  PI
2-19-80                 |P    E  I
2-20-80              P  |       E  I
2-21-80          P      |          E I
2-22-80        P        |           EI
2-23-80     P           |            *
2-24-80   P             |            *
2-25-80  P              |           IE
2-26-80  P              |          IE
2-27-80    P            |        IE
2-28-80      P          |       *
2-29-80         P       |     *
3-1-80              P   | EI
Continue (hit 'RETURN'), restart (enter 'R') or stop ('S')? _
```

With output option #1 selected, the program will respond with 24 days of biorhythm cycles. The user may stop, restart, or continue with the next 24 days. With option #2 (continuous mode) selected, the **CTRL C** keys are used to halt the program.

PROGRAM LISTING

```
100 REM  <BIORH>
110 REM Biorhythm calculator / plotter
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 CLEAR 200: LC = 0: CC = 15: PI = 3.14159
160 PRINT CHR$(12); "BIORHYTHMS": PRINT: PRINT
200 PRINT "Enter date of birth:"
210 GOSUB 2000: GOSUB 3000: NB = N
240 PRINT CHR$(10); "Enter chart starting date:"
250 GOSUB 2000: GOSUB 3000: NC = N
280 IF NC >= NB THEN 400
290 PRINT "Sorry! Chart date is before the date";
300 PRINT " of birth.  Try again!"
310 GOTO 200
400 PRINT CHR$(10); CHR$(10); "Output Options:"
420 PRINT TAB(5); "24 day cycle--enter '1'"
```

```
430 PRINT TAB(5); "Continuous plot--enter '2'"
440 INPUT "Your choice"; OP
450 IF OP < 1 OR OP > 2 THEN 400
460 GOSUB 4000
500 ND = NC - NB
510 CP = 23: GOSUB 5000: GOSUB 6000
520 CP = 28: GOSUB 5000: GOSUB 6000
530 CP = 33: GOSUB 5000: GOSUB 6000
600 GOSUB 7000
610 PRINT C$; TAB(9); L$; SPC(2);
615 IF LC = 0 OR LC > 9 THEN 630
620 ON LC GOTO 640,630,650,630,660,630,670,630,680
630 PRINT " ": GOTO 700
640 PRINT "P = Physical cycle": GOTO 700
650 PRINT "E = Emotional cycle": GOTO 700
660 PRINT "I = Intellectual cycle": GOTO 700
670 PRINT "* = Multiple points": GOTO 700
680 IF OP = 1 THEN PRINT: GOTO 700
690 PRINT "Hit 'CTRL C' to halt"
700 LC = LC + 1: IF LC < 24 THEN 750
710 LC = 0: IF OP = 2 THEN 750
720 PRINT "Continue (hit 'RETURN'), restart ";
725 INPUT "(enter 'R') or stop ('S')"; C$
730 IF C$ = "R" OR C$ = "r" THEN 100
735 IF C$ = "S" OR C$ = "s" THEN END
740 GOSUB 4000
750 GOSUB 8000: GOTO 500
1000 L$ = C$: FOR J = 2 TO 2*CC + 1
1010 L$ = L$ + C$: NEXT J
1020 RETURN
2000 PRINT TAB(5); "Month (1 to 12)";: INPUT M
2010 IF M <> INT(M) OR M < 1 OR M > 12 THEN 2000
2020 PRINT TAB(5); "Day (1 to 31)";: INPUT D
2030 IF D <> INT(D) OR D < 1 OR D >31 THEN 2020
2040 PRINT TAB(5); "Year";: INPUT Y
2050 IF Y <> INT(Y) OR Y < 0 THEN 2040
2060 IF Y > 99 THEN RETURN
2070 Y = Y + 1900: PRINT Y; "assumed.": RETURN
3000 LY = 0: RESTORE
3010 DATA 0, 31, 59, 90, 120, 151
3020 DATA 181, 212, 243, 273, 304, 334
3030 FOR J = 1 TO M: READ N: NEXT J
3040 N = N + 365*Y + D + 1
3050 N = N + INT(Y/4) - INT(Y/100) + INT(Y/400)
```



```
3060 IF Y <> INT(Y/4) * 4 THEN RETURN
3070 IF Y = INT(Y/400) * 400 THEN 3090
3080 IF Y = INT(Y/100) * 100 THEN RETURN
3090 LY = 1: IF M <= 2 THEN N = N - 1
3100 RETURN
4000 PRINT CHR$(12); SPC(20); "BIORHYTHM": PRINT
4010 PRINT "--DATE--"; TAB(12); "L o w";
4020 PRINT TAB(24); "0"; TAB(28); "H i g h"
4030 C$ = CHR$(176): GOSUB 1000
4040 PRINT TAB(9); L$: RETURN
5000 R = ND - INT(ND/CP) * CP: RETURN
6000 IF CP <> 23 THEN 6040
6010 C$ = " ": GOSUB 1000
6020 L$ = LEFT$(L$,CC) + CHR$(162) + RIGHT$(L$,CC)
6030 C$ = "P"
6040 IF CP = 28 THEN C$ = "E"
6050 IF CP = 33 THEN C$ = "I"
6060 W = SIN(2 * PI * (R/CP)) * CC
6070 W = INT(W + CC + 1.5)
6080 A$ = MID$(L$,W,1)
6090 IF A$="P" OR A$="E" OR A$="*" THEN C$="*"
6100 IF W <= 1 THEN 6140
6110 IF W >= 31 THEN 6150
6120 L$ = LEFT$(L$,W-1) + C$ + RIGHT$(L$,2*CC+1-W)
6130 RETURN
6140 L$ = C$ + RIGHT$(L$,2*CC): RETURN
6150 L$ = LEFT$(L$,2*CC) + C$: RETURN
7000 C$ = ""
7010 A$ = STR$(M): W = LEN(A$)
7020 C$ = C$ + RIGHT$(A$,W-1) + "-"
7030 A$ = STR$(D): W = LEN(A$)
7040 C$ = C$ + RIGHT$(A$,W-1) + "-"
7050 A$ = STR$(Y): W = LEN(A$)
7060 C$ = C$ + RIGHT$(A$,W-3): RETURN
8000 NC = NC + 1
8010 D = D + 1
8020 RESTORE
8030 FOR J = 1 TO M: READ W: NEXT J
8040 X = W: W = 365: IF M < 12 THEN READ W
8050 IF M = 2 AND LY = 1 THEN W = W + 1
8060 IF X + D <= W THEN RETURN
8070 D = 1: M = M + 1: IF M <= 12 THEN RETURN
8080 M = 1: Y = Y + 1: RETURN
```

EASY CHANGES

1. Want to see the number of days between any two dates? Insert this line:

   270 PRINT "DAYS="; NC – NB:END

   Then enter the earlier date as the birth date, and the later date as the start date for the chart. This will cause the program to display the difference in days and then end.
2. To alter the number of days of the chart shown on each screen, alter the 24 in line 700. You might prefer 14, for example.

MAIN ROUTINES

150 - 160	Initializes variables. Displays titles.
200 - 210	Asks for birth date and calculates the number of days since a fictional year zero.
240 - 250	Asks for start date for chart and calculates the day number.
280 - 310	Checks that chart date isn't sooner than birth date.
400 - 450	Gets choice of output options.
460	Displays heading at top of screen.
500	Determines number of days between birth and chart dates.
510 - 530	Plots points in string L$ for each of the three cycles.
600 - 690	Displays one line on the chart.
700 - 750	Adds one to chart date. Checks to see if the screen is full in the 24-day mode. Gets next input.
1000 - 1020	Subroutine to create a 31-character string.
2000 - 2070	Subroutine to ask operator for a month, day, and year.
3000 - 3100	Subroutine to find days from year zero to the input date.
4000 - 4040	Subroutine to clear screen and display headings.
5000	Subroutine to calculate remainder of ND/CP.
6000 - 6150	Subroutine to plot a point in L$ based on CP and R.

7000 - 7060 Subroutine to convert a numeric date into a string.
8000 - 8080 Adds one to chart date. Checks for end-of-month and end-of-year.

MAIN VARIABLES

LC	Counter for screen line number.
CC	Number of character spaces for one-half the chart.
PI	Pi.
NB	Days from year zero to birth.
NC	Days from year zero to chart starting date.
N	Day number calculated in subroutine.
J, W, X	Loop counters, work variables.
CP	Number of days in present biorhythm cycle (23, 28, or 33).
R	Remainder of ND/CP.
ND	Number of days between birth and current chart date.
L$	String with one line of chart.
A$, C$	String work variables.
M, D, Y	Month, day, year values.
LY	Leap year flag (set to 1 if true).
OP	Output option flag.

SUGGESTED PROJECTS

Investigate the biorhythms of some famous historical or athletic personalities. For example, are track and field athletes usually in the positive side of the physical cycle on the days that they set world records? Where was Lincoln in his emotional and intellectual cycles when he wrote "The Gettysburg Address"? Do a significant percentage of accidents befall people on critical days?

CHECKBOOK

PURPOSE

Many people consider the monthly ritual of balancing the checkbook to be an irritating and error-prone activity. Some people get confused and simply give up after the first try, while others give up the first time they cannot reconcile the bank statement with the checkbook. Fortunately, you have an advantage—your computer. This program takes you through the necessary steps to balance your checkbook, doing the arithmetic for you, of course.

HOW TO USE IT

The program starts off by giving you instructions to verify that the amount of each check or deposit is the same on the statement as it is in your checkbook. Sometimes the bank will make an error in reading the amount that you wrote on a check (especially if your handwriting is not too clear), and sometimes you will copy the amount incorrectly into your checkbook. While you are comparing these figures, make a check mark in your checkbook next to each check and deposit listed on the statement. A good system is to alternate the marks you use each month (maybe an "x" one month and a check mark the next) so you can easily see which checks and deposits came through on which statement.

Next the program asks for the ending balance shown on the bank statement. You are then asked for the *check number* (not the amount) of the most recent check shown on the statement. This will generally be the highest numbered check the bank has processed, unless you like to write checks out of sequence. Your account balance after this most recent check will be reconciled with the statement balance, so that is what the program asks for next—your checkbook balance after the most recent check on the statement.

The program must compensate for any differences between what your checkbook has in it prior to the most recent check and what the statement has on it. First, if you have any deposits that are not shown on the statement before the most recent check, you must enter them. Generally, there are none, so you just enter **END**.

Next you have to enter the amounts of any checks that have not yet "cleared" the bank and that are prior to the most recent check. Look in your checkbook for any checks that do not have your check mark next to them. Remember that some of these could be several months old.

Next you enter the amount of any service charges or debit memos that are on the statement, but which have not been shown in your checkbook prior to the most recent check. Typically, this is just a monthly service charge, but there might also be charges for printing new checks or some other adjustment that takes money away from you. Credit memos (which give money back to you) are not entered until later. Be sure to make an entry in your checkbook for each of these adjustments so that next month's statement will balance.

Finally, you are asked for any recent deposits or credit memos that are shown on the statement, but which follow the most recent cleared check in the checkbook. It's not unusual to find one or two of these, since deposits are often processed faster than checks.

Now comes the moment of truth. The program tells you whether or not you are in balance and displays the totals. If so, pack things up until next month's statement arrives.

If not, you have to figure out what is wrong. You have seven options of what to do next, including ones to allow review of the numbers you entered in case of a typing error. If you find an error, go back to the beginning and try again. Of course, if it is a simple error that precisely accounts for the amount by which you are out of balance, there is no need to go through the whole thing again.

If you entered everything correctly, the most likely cause of the out-of-balance condition is an arithmetic error in your checkbook. Look for errors in your addition and subtraction, with subtraction being the most likely culprit. This is especially likely if the amount of the error is a nice even number like one dollar or ten cents.

Another common error is accidentally adding the amount of a check in your checkbook instead of subtracting it. If you did this, your error will be twice the amount of the check (which makes it easy to find).

If this still does not explain the error, check to be sure you subtracted *last* month's service charge when you balanced your checkbook with the previous statement. And, of course, if you did not balance your checkbook last month, you cannot expect it to come out right this month.

The program has limitations of how many entries you can make in each category (checks outstanding, deposits outstanding, etc.), but these can be changed. See "Easy Changes" below.

NOTE: SEE DISCLAIMER IN FRONT PART OF BOOK.

SAMPLE RUN

```
CHECKBOOK BALANCER

The first step in balancing your checkbook is to
compare it with your most recent bank statement.
Compare each check and deposit shown on the state-
ment with the checkbook. Record all new service
charges or debits. Place a mark in the checkbook
next to each item on the statement.

What's the ending balance on this month's bank
statement (do not use a dollar sign)? 214.57

What is the number of the most recent check that is
shown on the statement? 1459

What balance does your checkbook show after check
1459 was subtracted? 107.12

Now I need a number of inputs. At the end of each
category, enter the word 'END' to go to the next
category (make sure the shift lock is down).

Enter the amount of each DEPOSIT or CREDIT MEMO
shown in your checkbook PRIOR TO check #1459 which
is NOT shown on this (or a prior) statement:
? END
```

```
Now enter the amount of each CHECK shown in your
checkbook PRIOR TO check #1459 which is NOT shown
on this (or a prior) statement:
? 23.89
? 84.81
? END

Enter the amounts of any SERVICE CHARGES or DEBIT
MEMOS that HAVE appeared on any statement, which
are AFTER check #1459 in your checkbook:
? 1.25
? END

Now enter the amount of each DEPOSIT or CREDIT
MEMO in the checkbook AFTER check #1459 which HAVE
appeared on this (or a prior) statement:
? END

CHECKBOOK BALANCER

Statement                      Checkbook
 balance:       $ 214.57        balance:        $ 1Ø7.12
  --plus--                       --plus--
Outstanding                    Outstanding
 deposits:      $ Ø             checks:         $ 1Ø8.7
  --plus--                       --plus--
Service                        Deposits
 charges:       $1.25           shown:          $ Ø

          TOTAL: $215.82                 TOTAL: $215.82

CONGRATULATIONS! It balances!

Options for next action:
     List (pre- # 1459) outstanding
      checks -- enter '1'
     List deposits not yet shown on
      any statement -- enter '2'
     List service charges -- enter '3'
     List (post - # 1459) recent
      deposits -- enter '4'
     Display category totals -- enter '5'
     Start over -- enter '6'
     Stop -- enter '7'
Your choice? 1

CHECKBOOK BALANCER

Checks outstanding:
$ 23.89
$ 84.81
Total $ 1Ø8.7

Options for next action:
   :
```

```
(Options menu listed again)
   :
Your choice? 2

CHECKBOOK BALANCER

Deposits outstanding:
None

Options for next action:
   :
(Options menu listed again)
   :
Your choice? 7
```

PROGRAM LISTING

```
100 REM <CBOOK>
110 REM Checkbook balancer
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Kugg & Phil Feldman
150 MCHECK = 10: MDEP = 5: MSERV = 5: MRDEP = 5
160 DIM CHECK(MCHECK), DEP(MDEP)
170 DIM SERV(MSERV), RDEP(MRDEP)
171 FOR J=1 TO MCHECK: CHECK(J) = 0: NEXT J
172 FOR J=1 TO MDEP: DEP(J) = 0: NEXT J
173 FOR J=1 TO MSERV: SERV(J) = 0: NEXT J
174 FOR J=1 TO MRDEP: RDEP(J) = 0: NEXT J
175 NCHECK = 0: NDEP = 0: NSERV = 0: NRDEP = 0
180 TCHECK = 0: TDEP = 0: TSERV = 0: TRDEP = 0
185 E$ = "Error -- please re-enter!": GOTO 200
190 PRINT CHR$(12); "CHECKBOOK BALANCER"
195 PRINT: PRINT: RETURN
200 GOSUB 190
210 PRINT "The first step in balancing your";
220 PRINT " checkbook is to compare it with"
230 PRINT " your most recent bank statement."
240 PRINT: PRINT "Compare each check and deposit";
250 PRINT " shown on the statement with the"
260 PRINT " checkbook.  Record all new ";
270 PRINT "service charges or debits.  Place"
280 PRINT " a mark in the checkbook next ";
290 PRINT "to each item on the statement."
320 PRINT: PRINT "What's the ending balance on ";
330 PRINT "this month's bank statement (do"
340 PRINT " not use a dollar sign)";: INPUT SBAL
```

```
350 PRINT: PRINT "What is the number of the ";
360 PRINT "most recent check that is shown on "
370 PRINT " the statement";: INPUT LASTC
380 PRINT: PRINT "What balance does your ";
390 PRINT "checkbook show after check"; LASTC
400 PRINT " was subtracted";: INPUT CBAL
410 GOSUB 190
420 PRINT "Now I need a number of inputs.  ";
430 PRINT "At the end of each category,"
440 PRINT " enter the word 'END' to go to ";
450 PRINT "the next category (make sure the"
460 PRINT " shift lock key is down)."; CHR$(10)
470 PRINT "Enter the amount of each DEPOSIT";
480 PRINT " or CREDIT MEMO shown in your "
490 PRINT " checkbook PRIOR TO check #"; LASTC;
500 PRINT "which is NOT shown on this"
510 PRINT " (or a prior) statement:"
520 INPUT R$: IF R$ = "END" THEN 580
530 IF VAL(R$) > 0 THEN 550
540 PRINT E$: GOTO 520
550 NDEP = NDEP + 1: DEP(NDEP) = VAL(R$)
560 TDEP = TDEP + DEP(NDEP)
565 IF NDEP < MDEP THEN 520
570 PRINT "Sorry--no more room!"
580 PRINT: PRINT "Now enter the amount of each ";
590 PRINT "CHECK shown in your checkbook "
600 PRINT " PRIOR TO check #"; LASTC; "which is";
610 PRINT " NOT shown on this (or a "
620 PRINT " prior) statement:"
630 INPUT R$: IF R$ = "END" THEN 700
640 IF VAL(R$) > 0 THEN 660
650 PRINT E$: GOTO 630
660 NCHECK = NCHECK + 1: CHECK(NCHECK) = VAL(R$)
670 TCHECK = TCHECK + CHECK(NCHECK)
680 IF NCHECK < MCHECK THEN 630
690 PRINT "Sorry--no more room!"
700 PRINT:PRINT "Enter the amounts of any SERVICE";
710 PRINT " CHARGES or DEBIT MEMOS that"
720 PRINT " HAVE appeared on any statement,";
730 PRINT " which are AFTER check #"; LASTC
740 PRINT " in your checkbook:"
750 INPUT R$: IF R$ = "END" THEN 820
760 IF VAL(R$) > 0 THEN 780
770 PRINT E$: GOTO 750
780 NSERV = NSERV + 1: SERV(NSERV) = VAL(R$)
```

```
790 TSERV = TSERV + SERV(NSERV)
800 IF NSERV < MSERV THEN 750
810 PRINT "Sorry--no more room!"
820 PRINT:PRINT "Now enter the amount of each ";
830 PRINT "DEPOSIT or CREDIT MEMO in the"
840 PRINT " checkbook AFTER check #"; LASTC;
850 PRINT "which HAVE appeared on this"
860 PRINT " (or a prior) statement:"
870 INPUT R$: IF R$ = "END" THEN 1000
880 IF VAL(R$) > 0 THEN 900
890 PRINT E$: GOTO 870
900 NRDEP = NRDEP + 1: RDEP(NRDEP) = VAL(R$)
910 TRDEP = TRDEP + RDEP(NRDEP)
920 IF NRDEP < MRDEP THEN 870
930 PRINT "Sorry--no more room!"
1000 GOSUB 190: GOSUB 2000
1010 W = SBAL+TDEP+TSERV - (CBAL+TCHECK+TRDEP)
1020 W = ABS(INT(100 * W)/100)
1030 IF W >= 0.01 THEN 1050
1040 PRINT "CONGRATULATIONS!  It balances!"
1045 GOTO 1080
1050 PRINT "Sorry, it doesn't balance.  The ";
1060 PRINT "two columns above should be"
1070 PRINT " equal, but they differ by $"; W
1080 PRINT: PRINT "Options for next action:"
1090 PRINT "    List (pre- #"; LASTC; CHR$(8);
1095 PRINT ") outstanding checks -- enter '1'"
1100 PRINT "    List deposits not yet shown on ";
1105 PRINT "any statement -- enter '2'"
1110 PRINT "    List service charges -- enter '3'"
1120 PRINT "    List (post- #"; LASTC; CHR$(8);
1125 PRINT ") recent deposits -- enter '4'"
1130 PRINT "    Show category totals -- enter '5'"
1140 PRINT "    Start over -- enter '6'"
1150 PRINT "    Stop -- enter '7'"
1160 INPUT "Your choice"; R
1170 ON R GOSUB 4100, 4200, 4300, 4400
1175 IF R=5 THEN GOSUB 190: GOSUB 2000: GOTO 1080
1180 IF R = 6 THEN 171
1190 IF R = 7 THEN END
1200 GOTO 1080
2000 PRINT "Statement balance: $"; SBAL; TAB(35);
2010 PRINT "Checkbook balance: $"; CBAL
2020 PRINT TAB(5); "--plus--"; TAB(40); "--plus--"
2030 PRINT "Outstanding"; TAB(35); "Outstanding"
```

```
2040 PRINT "         deposits: $"; TDEP; TAB(46);
2050 PRINT "checks: $"; TCHECK
2055 PRINT TAB(5); "--plus--"; TAB(40); "--plus--"
2060 PRINT "Service charges:   $"; TSERV; TAB(35);
2070 PRINT "Deposits shown:    $"; TRDEP; CHR$(10)
2080 PRINT TAB(13); "TOTAL: $";
2090 PRINT SBAL + TDEP + TSERV; TAB(47);
2100 PRINT "TOTAL: $"; CBAL + TCHECK + TRDEP
2110 PRINT: PRINT: RETURN
4100 GOSUB 190: PRINT "Checks outstanding:"
4110 IF NCHECK < 1 THEN PRINT "None": RETURN
4120 FOR J = 1 TO NCHECK: PRINT "$"; CHECK(J)
4125 NEXT J
4130 PRINT "Total $"; TCHECK: RETURN
4200 GOSUB 190: PRINT "Deposits outstanding:"
4210 IF NDEP < 1 THEN PRINT "None": RETURN
4220 FOR J = 1 TO NDEP: PRINT "$"; DEP(J): NEXT J
4230 PRINT "Total $"; TDEP: RETURN
4300 GOSUB 190: PRINT "Service charges:"
4310 IF NSERV < 1 THEN PRINT "None": RETURN
4320 FOR J = 1 TO NSERV: PRINT "$"; SERV(J): NEXT J
4330 PRINT "Total $"; TSERV: RETURN
4400 GOSUB 190: PRINT "Deposits after check #";
4405 PRINT LASTC; CHR$(8); ":"
4410 IF NRDEP < 1 THEN PRINT "None": RETURN
4420 FOR J = 1 TO NRDEP: PRINT "$"; RDEP(J): NEXT J
4430 PRINT "Total $"; TRDEP: RETURN
```

EASY CHANGES

Change the limitations of how many entries you can make in each category. Line 150 establishes these limits. If you have more than ten checks outstanding at some time, change the value of MCHECK to 20, for example. The other three variables can also be changed if you anticipate needing more than five entries. They are: the number of deposits outstanding (MDEP), the number of service charges and debit memos (MSERV), and the number of recent deposits and credit memos (MRDEP).

MAIN ROUTINES

150-290	Initializes variables and displays first instructions.
320-370	Gets most recent check number and statement balance.

380-400	Gets checkbook balance after most recent check number.
410-570	Gets outstanding deposits.
580-690	Gets outstanding checks.
700-810	Gets service charges and debit memos.
820-930	Gets recent deposits and credit memos.
1000-1070	Does balancing calculation. Displays it.
1080-1200	Asks for next action. Goes to appropriate subroutine or stops or restarts program.
2000-2110	Subroutine to print balancing data.
4100-4130	Subroutine to display checks outstanding.
4200-4230	Subroutine to display deposits outstanding.
4300-4330	Subroutine to display service charges and debit memos.
4400-4430	Subroutine to display recent deposits.

MAIN VARIABLES

MCHECK	Maximum number of checks outstanding.
MDEP	Maximum number of deposits outstanding.
MSERV	Maximum number of service charges, debit memos.
MRDEP	Maximum number of recent deposits, credit memos.
CHECK	Array for checks outstanding.
DEP	Array for deposits outstanding.
SERV	Array for service charges and debit memos.
RDEP	Array for recent deposits and credit memos.
TCHECK	Total of checks outstanding.
TDEP	Total of deposits outstanding.
TSERV	Total of service charges and debit memos.
TRDEP	Total of recent deposits and credit memos.
NCHECK	Number of checks outstanding.
NDEP	Number of deposits outstanding.
NSERV	Number of service charges and debit memos.
NRDEP	Number of recent deposits and credit memos.
E$	Error message.
SBAL	Statement balance.
LASTC	Number of last check on statement.
CBAL	Checkbook balance after last check on statement.
R$	Reply from operator.
W	Amount by which checkbook is out of balance.
R	Numeric value of reply for next action.

SUGGESTED PROJECTS

1. Add more informative messages and a more complete introduction to make the program a tutorial for someone who has never balanced a checkbook before.
2. Allow the operator to modify any entries that have been discovered to be in error. This could be done by adding another option to the menu list, which would then ask the operator which category to change. This would allow the operator to correct an error without having to reenter everything from the beginning.
3. If the checkbook is out of balance, have the program do an analysis (as suggested in the "How To Use It" section) and suggest the most likely errors that might have caused the condition.
4. Allow the operator to find arithmetic errors in the checkbook. Ask for the starting balance, then ask for each check or deposit amount. Add or subtract, depending on which type the operator indicates. Display the new balance after each entry so the operator can compare with the checkbook entry.

PICKS

PURPOSE

"Decisions, decisions!" How many times have you uttered this lament when confronted by a difficult choice? Wouldn't a trusty advisor be helpful on such occasions? Well, you now have one–your Sorcerer computer, of course.

This program can help you make decisions involving the selection of one alternative from several choices. It works by prying relevant information from you and then organizing it in a meaningful, quantitative manner. Your best choice will be indicated and all of the possibilities given a relative rating.

You can use the program for a wide variety of decisions. It can help with things like choosing the best stereo system, saying yes or no to a job or business offer, or selecting the best course of action for the future. Everything is personalized to your individual decision.

HOW TO USE IT

The first thing the program does is ask you to categorize the decision at hand into one of three types:

1) Choosing an item (or thing);
2) Choosing a course of action; or
3) Making a yes or no decision.

You simply input the number of the type of decision facing you. If you are choosing an item, you will be asked what type of item it is.

If the decision is either of the first two types, you must next enter a list of all the possibilities under consideration. A question mark will prompt you for each one. When the list is complete, type **END** in response to the last question mark. You must, of course, enter at least two possibilities. (We hope you don't have trouble making decisions from only one possibility!) After the list is finished, it will be redisplayed so that you can verify that it is correct. If not, you must reenter it.

Now you must think of the different factors that are important to you in making your decision. For example, location, cost, and quality of education might govern the decision of which college to attend. For a refrigerator purchase, the factors might be things like price, size, reliability, and warranty. In any case, you will be prompted for your list with a succession of question marks. Each factor is to be entered one at a time with the word **END** used to terminate the list. When complete, the list will be redisplayed. You must now decide which single factor is the most important and input its number.

The program now asks you to rate the importance of each of the other factors relative to the most important one. This is done by first assigning a value of 10 to the main factor. Then you must assign a value from 0 - 10 to each of the other factors. These numbers reflect your assessment of each factor's relative importance as compared to the main one. A value of 10 means it is just as important; lesser values indicate how much less importance you place on it.

Now you must rate the decision possibilities with respect to each of the importance factors. Each importance factor will be treated separately. Considering *only* that importance factor, you must rate how each decision possibility stacks up. The program first assigns a value of 10 to one of the decision possibilities. Then you must assign a relative number (lower, higher, or equal to 10) to each of the other decision possibilities.

An example might alleviate possible confusion here. Suppose you are trying to decide whether to get a dog, cat, or canary for a pet. Affection is one of your importance factors. The program assigns a value of 10 to the cat. Considering affection *only*, you might assign a value of 20 to the dog and 6.5 to the canary. This means *you* consider a dog twice as affectionate as a cat but a canary only about two-thirds as affectionate as a cat. (No slighting of bird lovers is intended here, of course. Your actual ratings may be entirely different.)

Armed with all this information, the program will now determine which choice seems best for you. The various possibilities are listed in order of ranking. Alongside each one is a relative rating with the best choice being normalized to a value of 100.

Of course, PICKS should not be used as a substitute for good, clear thinking. However, it can often provide valuable insights. You might find one alternative coming out surprisingly low or high. A trend may become obvious when the program is rerun with improved data. At least, it may help you think about decisions systematically and honestly.

SAMPLE RUN

```
DECISION-SELECTION ROUTINE

The Sorcerer of Exidy is here!

I can help you conjure up a decision or materialize
a selection from various alternatives. Please provide
me with the data I request.

******************************************************

Which of the following best describes the type of
decision facing you:
   (1) Selecting an item from a group of
       possibilities.
   (2) Selecting a course of action from a
       group of possible actions.
   (3) Making a 'Yes' or 'No' decision.
Which number? 1

What type of item are you selecting? vacation

I need a list of each vacation under consideration.
Please input them, one at a time in any order.
Input 'END' to indicate the end of the list.

Alternative # 1? mountain camping

Alternative # 2? African safari

Alternative # 3? trip to D.C.

Alternative # 4? end

OK, here's the list you've given me:
   # 1--mountain camping
   # 2--African safari
   # 3--trip to D.C.
```

```
Is this list correct (enter 'Y' or 'N')? Y

Think of the different factors that are
important to you in choosing the best vacation.
Input them one at a time, in any order. Enter
'END' to mark the end of the list.

Factor # 1? relaxation

Factor # 2? cost

Factor # 3? change of pace

Factor # 4? end

OK, here's the list you've given me:
   # 1--relaxation
   # 2--cost
   # 3--change of pace
Is this list correct (enter 'Y' or 'N')? Y

Now, input the item number of the factor which
is most important to you? 2

Assume that these factors are ranked on a scale
of Ø-1Ø, based on importance to you. I'll set
cost at a value of 1Ø, since you rated it as
most important.

On this scale, how do these rank?

Factor # 1--relaxation? 3

Factor # 3--change of pace? 9

Each vacation must now be compared, with respect
to each importance factor. We'll rate each of
these choices separately, for one factor at a time.

I'll arbitrarily assign a value of 1Ø to the
first alternative.  Each other alternative will
then be assigned a value either higher or lower
than 1Ø, depending on whether you feel it is
better or worse than mountain camping.

----------------------
Considering only the factor of relaxation and
assigning a value of 1Ø to mountain camping,
what value (zero or greater) do you assign to:

Alternative # 2--African safari? 1
Alternative # 3--trip to D.C.? 8
----------------------
Considering only the factor of cost and
```

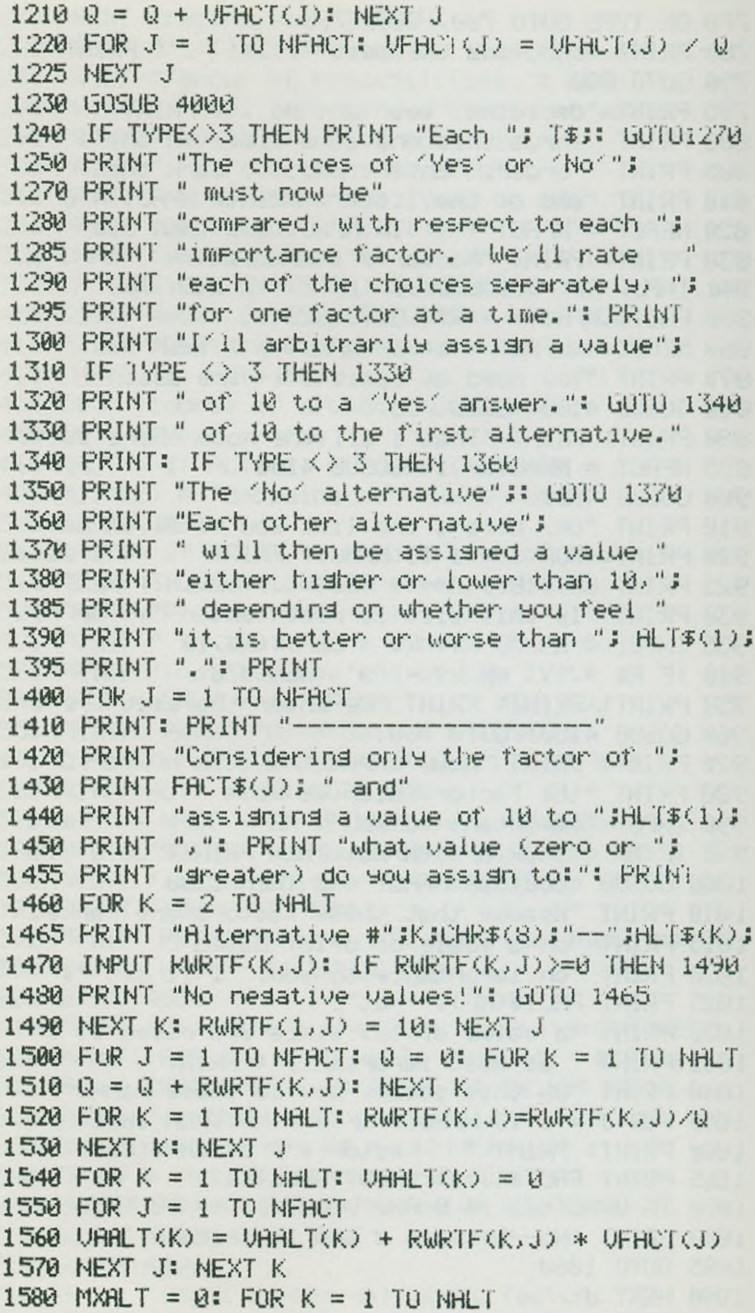
```
assigning a value of 10 to mountain camping,
what value (zero or greater) do you assign to:

Alternative # 2--African safari? 0
Alternative # 3--trip to D.C.? 15

----------------------
Considering only the factor of change of pace
and assigning a value of 10 to mountain camping,
what value (zero or greater) do you assign to:

Alternative # 2--African safari? 75
Alternative # 3--trip to D.C.? 4

The winner is African safari!
However, it was very close.

Here is the final list, in order. The winning
alternative has been given a value of 100, and
the others are ranked accordingly.

----------------------

100            African safari

99.0376        trip to D.C.

85.1207        mountain camping
```

PROGRAM LISTING

```
100 REM <PICKS>
110 REM Decision / selection
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 CLEAR 2000: SIZE = 10
160 DIM ALT$(SIZE), FACT$(SIZE), VFACT(SIZE)
170 DIM RWRTF(SIZE,SIZE), VAALT(SIZE), RALT(SIZE)
180 E1$="END": E2$="end": E3$="End": W$ = CHR$(132)
200 GOSUB 4000
210 PRINT "The Sorcerer of Exidy is here!": PRINT
220 PRINT "I can help you conjure up a deci";
225 PRINT "sion or materialize a selection "
230 PRINT "from various alternatives.  Please ";
240 PRINT "provide the data I request.": PRINT
250 FOR J = 1 TO 32: PRINT W$;" ";: NEXT J: PRINT
255 PRINT: PRINT
260 PRINT "Which of the following best ";
265 PRINT "describes the type of decision "
```

```
270 PRINT "facing you:": PRINT
280 PRINT "    (1) Selecting an item from a";
285 PRINT " group of possibilities.": PRINT
290 PRINT "    (2) Selecting a course of action ";
295 PRINT "from a group of possible"
300 PRINT "         actions.": PRINT
310 PRINT "    (3) Making a 'Yes' or 'No' ";
315 PRINT "decision.": PRINT
320 INPUT "Which number (1, 2, or 3)"; TYPE
330 IF TYPE < 1 OR TYPE > 3 THEN 320
400 GOSUB 4000: ON TYPE GOTO 410, 430, 450
410 PRINT "What type of item are you selecting";
420 INPUT T$: GOTO 500
430 T$ = "course of action": GOTO 500
450 T$ = "'Yes' or 'No'"
460 NALT = 2: ALT$(1) = "deciding yes"
470 ALT$(2) = "deciding no": GOTO 750
500 GOSUB 4000
510 NALT = 0: PRINT "I need a list of each "; T$;
515 PRINT " under consideration.": PRINT
520 PRINT "Please input them, one at a time";
525 PRINT " in any order.  Input 'END' to  "
530 PRINT "indicate the end of the list:  "
540 NALT = NALT + 1: IF NALT > 9 THEN 592
550 PRINT: PRINT "Alternative #"; NALT;
560 INPUT R$: GOSUB 5000: IF E = 1 THEN 570
565 ALT$(NALT) = R$: GOTO 540
570 NALT = NALT - 1: IF NALT >= 2 THEN 600
580 PRINT "Sorry, you must have at least two";
590 PRINT " alternatives--try again!"
591 GOSUB 4100: GOTO 500
592 PRINT: PRINT "That's all the room there is!"
595 NALT = NALT - 1: GOSUB 4100
600 GOSUB 4000
610 PRINT "OK, here's the list you've given me:"
620 PRINT: FOR J = 1 TO NALT: PRINT "   #"; J;
625 PRINT CHR$(8); "--"; ALT$(J): PRINT: NEXT J
630 PRINT "is this list correct (enter 'Y' or ";
635 INPUT "'N')"; R$: R$ = LEFT$(R$,1)
640 IF R$ = "Y" OR R$ ="y" THEN 750
650 PRINT: PRINT: PRINT "Re-enter the list . . ."
660 GOSUB 4100: GOTO 500
750 GOSUB 4000
760 PRINT "Think of the different factors ";
765 PRINT "that are important to you in"
```

```
770 ON TYPE GOTO 780, 780, 795
780 PRINT "choosing the best "; T$; ".": PRINT
790 GOTO 800
795 PRINT "deciding 'Yes' or 'No'.": PRINT
800 PRINT "Input them one at a time, in any";
805 PRINT " order.  Enter 'END' to mark the"
810 PRINT "end of the list:": PRINT: NFACT = 0
820 NFACT = NFACT + 1: IF NFACT > 9 THEN 890
830 PRINT: PRINT "Factor #"; NFACT;
840 INPUT R$: GOSUB 5000: IF E = 1 THEN 860
850 FACT$(NFACT) = R$: GOTO 820
860 NFACT = NFACT - 1: IF NFACT > 1 THEN 900
870 PRINT "You need at least one--try again!"
880 GOSUB 4100: GOTO 750
890 PRINT: PRINT "That's all the room there is!"
895 NFACT = NFACT - 1: GOSUB 4100
900 GOSUB 4000
910 PRINT "OK, here's the list you've given me:"
920 PRINT: FOR J = 1 TO NFACT: PRINT "   #"; J;
925 PRINT CHR$(8); "--"; FACT$(J): PRINT: NEXT J
930 PRINT "Is this list correct (enter 'Y' or ";
935 INPUT "'N')"; R$: R$ = LEFT$(R$,1)
940 IF R$ = "Y" OR R$ = "y" THEN 970
950 PRINT: PRINT: PRINT "Re-enter the list . . ."
960 GOSUB 4100: GOTO 750
970 PRINT: PRINT "Now, input the item number of ";
980 PRINT "the factor which is most "
990 INPUT "important to you"; MF
995 IF MF < 1 OR MF > NFACT THEN PRINT: GOTO 910
1000 GOSUB 4000: IF NFACT < 2 THEN 1200
1010 PRINT "Assume that these factors are ranked ";
1015 PRINT "on a scale of 0-10, based "
1020 PRINT "on importance to you.  I'll set ";
1025 PRINT FACT$(MF); " at"
1030 PRINT "a value of 10, since you rated it";
1035 PRINT " as most important.": PRINT
1040 PRINT "On this scale, how do these rank?"
1050 FOR J = 1 TO NFACT: IF J = MF THEN 1090
1060 PRINT: PRINT "   Factor #"; J; CHR$(8); "--";
1065 PRINT FACT$(J);: INPUT VFACT(J)
1070 IF VFACT(J) >= 0 AND VFACT(J) <= 10 THEN 1090
1080 PRINT "No--between 0 and 10, please!"
1085 GOTO 1060
1090 NEXT J
1200 VFACT(MF) = 10: Q = 0: FOR J = 1 TO NFACT
```



```
1210 Q = Q + VFACT(J): NEXT J
1220 FOR J = 1 TO NFACT: VFACT(J) = VFACT(J) / Q
1225 NEXT J
1230 GOSUB 4000
1240 IF TYPE<>3 THEN PRINT "Each "; T$:: GOTO1270
1250 PRINT "The choices of 'Yes' or 'No'";
1270 PRINT " must now be"
1280 PRINT "compared, with respect to each ";
1285 PRINT "importance factor.  We'll rate  "
1290 PRINT "each of the choices separately, ";
1295 PRINT "for one factor at a time.": PRINT
1300 PRINT "I'll arbitrarily assign a value";
1310 IF TYPE <> 3 THEN 1330
1320 PRINT " of 10 to a 'Yes' answer.": GOTO 1340
1330 PRINT " of 10 to the first alternative."
1340 PRINT: IF TYPE <> 3 THEN 1360
1350 PRINT "The 'No' alternative";: GOTO 1370
1360 PRINT "Each other alternative";
1370 PRINT " will then be assigned a value "
1380 PRINT "either higher or lower than 10,";
1385 PRINT " depending on whether you feel "
1390 PRINT "it is better or worse than "; ALT$(1);
1395 PRINT ".": PRINT
1400 FOR J = 1 TO NFACT
1410 PRINT: PRINT "--------------------"
1420 PRINT "Considering only the factor of ";
1430 PRINT FACT$(J); " and"
1440 PRINT "assigning a value of 10 to ";ALT$(1);
1450 PRINT ",": PRINT "what value (zero or ";
1455 PRINT "greater) do you assign to:": PRINT
1460 FOR K = 2 TO NALT
1465 PRINT "Alternative #";K;CHR$(8);"--";ALT$(K);
1470 INPUT RWRTF(K,J): IF RWRTF(K,J)>=0 THEN 1490
1480 PRINT "No negative values!": GOTO 1465
1490 NEXT K: RWRTF(1,J) = 10: NEXT J
1500 FOR J = 1 TO NFACT: Q = 0: FOR K = 1 TO NALT
1510 Q = Q + RWRTF(K,J): NEXT K
1520 FOR K = 1 TO NALT: RWRTF(K,J)=RWRTF(K,J)/Q
1530 NEXT K: NEXT J
1540 FOR K = 1 TO NALT: VAALT(K) = 0
1550 FOR J = 1 TO NFACT
1560 VAALT(K) = VAALT(K) + RWRTF(K,J) * VFACT(J)
1570 NEXT J: NEXT K
1580 MXALT = 0: FOR K = 1 TO NALT
1590 IF VAALT(K) > MXALT THEN MXALT = VAALT(K)
```

```
1600 NEXT K: FOR K = 1 TO NALT
1610 VAALT(K) = VAALT(K) * 100 / MXALT: NEXT K
1620 FOR K = 1 TO NALT: RALT(K) = K: NEXT K
1630 FOR K = 1 TO NALT: FOR J = 1 TO NALT - 1
1640 X = RALT(J): Y = RALT(J+1)
1650 IF VAALT(X) > VAALT(Y) THEN 1670
1660 RALT(J+1) = X: RALT(J) = Y
1670 NEXT J: NEXT K
1680 X = RALT(1): Y = RALT(2)
1690 DF = VAALT(X) - VAALT(Y)
1700 GOSUB 4000: IF DF > 0 THEN 1710
1705 PRINT "It's a tie!": GOTO 1800
1710 PRINT "The winner is "; ALT$(X); "!"
1720 IF DF >= 5 THEN 1740
1730 PRINT "However, it was very close.":GOTO1800
1740 IF DF >= 10 THEN 1760
1750 PRINT "It was fairly close.": GOTO 1800
1760 IF DF >= 20 THEN 1780
1770 PRINT "It won by a fair amount.": GOTO 1800
1780 PRINT "It won quite decisively."
1800 PRINT: PRINT "Here is the final list, in ";
1810 PRINT "order.  The winning alternative has"
1820 PRINT "been given a value of 100, and the";
1830 PRINT " others are ranked accordingly."
1840 PRINT: PRINT "--------------------": PRINT
1850 FOR J = 1 TO NALT: Q = RALT(J)
1860 PRINT VAALT(Q); TAB(16); ALT$(Q): PRINT
1870 NEXT J: END
4000 FOR J = 1 TO 1000: NEXT J
4010 PRINT CHR$(12), "DECISION-SELECTION ROUTINE"
4020 PRINT: PRINT: RETURN
4100 FOR J = 1 TO 2500: NEXT J: RETURN
5000 E = 0: IF R$ = E1$ THEN E = 1: RETURN
5010 IF R$ = E2$ OR R$ = E3$ THEN E = 1: RETURN
5020 RETURN
```

EASY CHANGES

1. The word "END" is used to flag the termination of various input lists. If you wish to use something else (because of conflicts with items on the list), change the definitions in line 180. For example, to use the word "DONE," change the string in line 180 accordingly.
2. Lines 4000 and 4100 contain a timing delay used regularly in

the program. If things seem to change too fast, you can make the delay longer. Try

```
4100 FOR J=1 TO 5000:NEXT J:RETURN
```

3. The program can currently accept up to nine decision alternatives and/or nine importance factors. If you need more, increase the value of SIZE in line 150. Each dimension value is one more than the number the program will actually allow. Thus, to use 14 values, line 150 should be:

```
150 CLEAR 2000:SIZE=15
```

MAIN ROUTINES

150-180	Initializes and dimensions variables.
200-330	Determines category of decision.
400-470	Gets or sets T$.
500-660	Gets list of possible alternatives from user.
750-960	Gets list of importance factors from user.
970-1225	User rates each importance factor.
1230-1490	User rates the decision alternatives with respect to each importance factor.
1500-1570	Evaluates the various alternatives.
1580-1690	Sorts alternatives into their relative ranking.
1700-1870	Displays results.
4000-4020	Subroutine to clear screen and display header.
4100	Time wasting subroutine.
5000-5020	Sets end-of-input flag.

MAIN VARIABLES

NALT	Number of decision alternatives.
ALT$	String array of the decision alternatives.
NFACT	Number of importance factors.
FACT$	String array of the importance factors.
VFACT	Array of the relative values of each importance factor.
MF	Index number of most important factor.
RWRTF	Array of relative values of each alternative with respect to each importance factor.
TYPE	Decision category (1 = item, 2 = course of action, 3 = yes or no).
T$	String name of decision category.

E1$, E2$, E3$	Strings to signal the end of an input data list.
J,K	Loop indices.
R$	User reply string.
Q, X, Y	Work variables.
W$	Work string variable.
E	End-of-input flag.
VAALT	Array of each alternative's value.
MXALT	Maximum value of all alternatives.
DF	Rating difference between best two alternatives.
RALT	Array of the relative rankings of each alternative.

SUGGESTED PROJECTS

1. Allow the user to review his numerical input and modify it if desired.
2. Insights into a decision can often be gained by a sensitivity analysis. This involves running the program a number of times for the same decision. Each time, one input value is changed (usually the one you are least confident about). By seeing how the results change, you can determine which factors are the most important. Currently, this requires a complete rerunning of the program each time. Modify the program to allow a change of input after the regular output is produced. Then recalculate the results based on the new values. (Note that many arrays are "clobbered" once all the input is given. This modification will require saving the original input in new arrays so that it can be reviewed later.)

LOAN

PURPOSE

One of the most frustrating things about borrowing money from a bank (or credit union or Savings and Loan) is that it's not easy to fully evaluate your options. When you are borrowing from a credit union to buy a new car, you might have the choice of a 36- or a 48-month repayment period. When buying a house, you can sometimes get a slightly lower interest rate for your loan if you can come up with a larger down payment. Which option is best for you? How will the monthly payment be affected? Will there be much difference in how fast the principal of the loan decreases? How much of each payment will be for interest, which is tax-deductible?

You need to know the answers to all these questions to make the best decision. This program gives you the information you need.

HOW TO USE IT

First the program asks for the size of the loan. Use whole dollar amounts only, with a maximum of a million dollars (if you need more than that, you can hire a professional investment counselor!). Next, enter the annual interest rate, as a percentage; if the rate is 12%, enter **12**. The final input describing the desired loan is the term, in months; for example, a 10-year loan would have a term of 120.

This data is displayed, along with the uniform monthly payment required to pay off the loan over the stated term. At this point, the program will offer four options.

First, you may obtain a month-by-month breakdown, showing the state of the loan after each payment. The six columns of data shown for each month are the payment (month) number, remaining balance, and the monthly and accumulated totals of both interest and principal.

The second option overrides the calculated uniform payment. It is common, particularly on second mortgages, to make small monthly payments with a large "balloon" payment in the last month. You can use this second option to try various monthly payments to see how the final balloon is changed. Similarly, you can see the effects of excess payments on the term of the loan. After overriding the monthly payment with this option, rerun the first option to get a monthly analysis and totals.

The third option begins the program again, allowing input of new terms, principal, and interest. The fourth option ends the program.

There may be some variation between the program's calculations and those of your banker. Since the Sorcerer can only deal with a maximum of six digits, totals are stored as whole dollars while monthly values are accurate to the nearest cent. Thus, some minor rounding errors may occur.

NOTE: SEE DISCLAIMER AT FRONT OF BOOK.

SAMPLE RUN

```
LOAN CALCULATOR

What is the principal amount ($)? 4500

Annual interest (percent)? 10.25

Term of loan (months)? 20

For $ 4500 at 10.25% over 20 months:
The monthly payment is $ 245.72

Options for next action:
   Monthly analysis--enter '1'
   Change monthly payment--enter '2'
   Start over--enter '3'
   End program--enter '4'
Your choice? 1
```

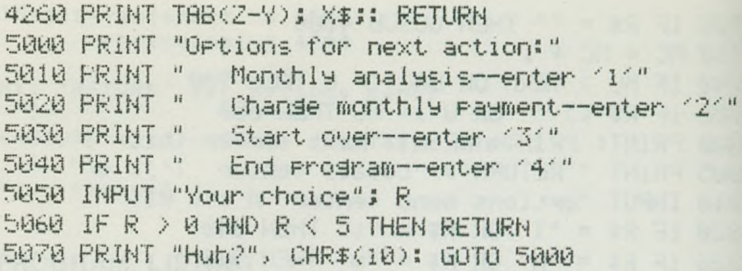
```
For $ 4500 at 10.25% over 20 months:
Months 1 thru 16

        --Monthly---  --To Date---  Remaining
Month   Int.  Princ.  Int.  Princ.  Balance
    1  38.44  207.28   38     207       4293
    2  36.67  209.05   75     416       4084
    3  34.88  210.84  110     627       3873
    :
    :
   16  10.22  235.50  393    3540        960

Next screen (hit 'RETURN'), totals (enter 'T').
or options menu (enter 'M')?_ (RETURN hit)

For $ 4500 at 10.25% over 20 months:
Months 17 thru 20

        --Monthly---  --To Date---  Remaining
Month   Int.  Princ.  Int.  Princ.  Balance
   17   8.20  237.52  401    3778        722
   18   6.17  239.55  407    4018        482
   19   4.12  241.60  411    4260        240
   20   2.05  240.00  413    4500          0

Payment--monthly  $ 245.72 (for 19 months)
       --final    $ 242.05
Totals--interest  $ 413
      --principal $4500

Hit 'RETURN' for options menu?_
  :
  :
(RETURN hit, options displayed again)
  :
Your choice? 2

Current monthly payment is $ 245.72
New monthly payment? 225.00
  :
  :
(Options menu displayed again)
  :
Your choice? 1

        --Monthly---  --To Date---  Remaining
Month   Int.  Princ.  Int.  Princ.  Balance
    1  38.44  186.56   38     187       4313
    2  36.84  188.16   75     375       4125
    :
    :
   19   7.57  217.43  444    3831        669
   20   5.71  669.00  450    4500          0

Payment--monthly  $ 225 (for 19 months)
       --final    $ 674.71
```

```
Totals--interest  $ 450
     --principal  $ 4500

Hit 'RETURN' for options menu?
  :
(RETURN hit, options displayed again)
  :
Your choice? 4
```

PROGRAM LISTING

```
100 REM <LOAN>
110 REM Loan calculator
120 PFM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 PFM by Kevin McCabe, Tom Russ & Phil Feldman
200 PRINT CHR$(12);"LOAN CALCULATOR":PRINT:PRINT
210 INPUT "What is the principal amount ($)"; PAM
220 PAM = INT(PAM): IF PAM>0 AND PAM<1E6 THEN 240
230 PRINT "Between 0 and $100,000 only!":GOTO 210
240 PRINT: INPUT "Annual interest (percent)"; APR
250 IF APR < 0 THEN PRINT "Huh?": GOTO 240
260 PRINT: INPUT "Term of loan (months)"; TERM
270 TERM = INT(TERM): IF TERM > 0 THEN 300
280 PRINT "huh?": GOTO 260
300 MR = APR / 1200: W = (1 + MR) ↑ TERM
310 PY = INT(100 * PAM * MR * W / (W-1) + 0.5)
330 GOSUB 2000
340 PRINT "The monthly payment is $";
350 PRINT PY/100: PRINT: PRINT
400 GOSUB 5000
410 ON R GOTO 600, 500, 200: END
500 PRINT: PRINT "Current monthly payment is $";
510 PRINT PY/100: INPUT "New monthly payment"; PY
520 PY = INT(100 * PY + 0.5): GOTO 330
600 MC = 1: TI = 0: TP = 0: BAL = PAM: PN = PY
620 W = MC + 15: IF W > TERM THEN W = TERM
630 GOSUB 2000: PRINT "Months"; MC; "thru"; W
640 PRINT: PRINT: GOSUB 3000: R$ =""
650 MI = INT(100 * BAL * MR + 0.5)
660 IF PY<=100*BAL + MI AND MC<>TERM THEN 680
670 PY = 100 * BAL + MI
680 MP = PY - MI: TI = TI + INT(MI/100 + 0.5)
700 TP = TP + INT(MP/100 + 0.5)
710 BAL = BAL - INT(MP/100 + 0.5)
```

```
720 IF R$ = "" THEN GOSUB 4000
730 MC = MC + 1
740 IF MC > TERM OR BAL <= 0 THEN 900
750 IF R$ <> "" OR W >= MC THEN 650
800 PRINT: PRINT: PRINT "Next screen (hit ";
805 PRINT "'RETURN'), totals (enter 'T'), or"
810 INPUT "options menu (enter 'M')"; R$
820 IF R$ = "T" OR R$ = "t" THEN 840
825 IF R$ = "M" OR R$ = "m" THEN 980
830 R$ = "": GOTO 620
840 PRINT "OK--Just a second . . ."
845 IF MC <= TERM OR BAL > 0 THEN 650
900 PRINT: PRINT: PRINT "Payment--monthly  $";
920 PRINT PN/100; "(for"; MC - 2; "months)"
930 PRINT "        --final    $"; PY/100
940 PRINT "Totals--interest  $"; TI
950 PRINT "       --principal $"; TP
960 R$ = "T": PY = PN: PRINT: PRINT
970 INPUT "Hit 'RETURN' for options menu"; R$
980 PRINT: PRINT: GOTO 400
2000 PRINT CHR$(12); "LOAN CALCULATOR"; CHR$(12)
2010 PRINT: PRINT "For $"; PAM; "at"; APR;
2020 PRINT CHR$(8); "% over"; TERM; "months:"
2030 RETURN
3000 PRINT SPC(8); "--Monthly Payment---"; SPC(3);
3010 PRINT "----Paid To Date----"; SPC(3);
3020 PRINT "Remaining": PRINT "Month"; SPC(3);
3030 PRINT "Interest  Principal  Interest  ";
3040 PRINT "Principal  Balance": RETURN
4000 X = MC: Z = 5: GOSUB 4100
4010 X = MI/100: Z = 16: GOSUB 4200
4020 X = MP/100: Z = 28: GOSUB 4200
4030 X = TI: Z = 39: GOSUB 4100
4040 X = TP: Z = 51: GOSUB 4100
4050 X = BAL: Z=63: GOSUB 4100
4060 PRINT: RETURN
4100 Y = LEN(STR$(X)): PRINT TAB(Z-Y); X;
4110 RETURN
4200 X$ = STR$(X): Y = LEN(X$) - 1
4210 X$ = RIGHT$(X$,Y)
4215 X = INT(100 * VAL(X$)) / 100
4220 IF X <> INT(X) THEN 4240
4230 X$ = X$ + ".00": Y = Y + 3: GOTO 4260
4240 IF MID$(X$,Y-1,1) <> "." THEN 4260
4250 X$ = X$ + "0": Y = Y + 1
```



```
4260 PRINT TAB(Z-Y); X$;: RETURN
5000 PRINT "Options for next action:"
5010 PRINT "   Monthly analysis--enter '1'"
5020 PRINT "   Change monthly payment--enter '2'"
5030 PRINT "   Start over--enter '3'"
5040 PRINT "   End program--enter '4'"
5050 INPUT "Your choice"; R
5060 IF R > 0 AND R < 5 THEN RETURN
5070 PRINT "Huh?", CHR$(10): GOTO 5000
```

EASY CHANGES

1. The number of lines of data that are displayed on each screen when getting a monthly analysis can be changed by altering the constant 15 in statement 620. You might want only 12 payments shown at a time so you can see a year-by-year analysis.
2. To include the monthly payment in the heading at the top each screen of the monthly analysis, insert the following line:

   635 PRINT "Monthly payment is $"; PY/100

MAIN ROUTINES

200-280	Displays title. Gets loan information.
300-350	Calculates and displays monthly payment.
400-410	Asks for next action. Goes to corresponding routine.
500-520	Gets override for monthly payment.
600-980	Calculates and displays monthly analysis and totals.
2000-2030	Subroutine to clear screen and display data about the loan at the top.
3000-3040	Headings subroutine.
4000-4260	Subroutine to convert numeric amount to string with aligned decimal point.
5000-5070	Options menu subroutine.

MAIN VARIABLES

PAM	Amount of loan.
APR	Interest rate (percentage).
TERM	Length of loan (number of months).

MR	Monthly interest rate (not percentage).
W	Work variable.
PY	Monthly payment.
PN	Saves monthly payment.
R	Choice of next action.
BAL	Remaining balance of loan.
TI	Total interest to date (dollars).
TP	Total principal to date (dollars).
J	Work variable for loops.
MI	Monthly interest (cents).
X$	Work string variable for creating numbers with decimal point alignment.
MP	Monthly principal (cents).
X, Y, Z	Work variables for printing X$.

SUGGESTED PROJECTS

1. Display a more comprehensive analysis of the loan along with the final totals. Show the ratio of total payments to the amount of the loan, for example.
2. Modify the program to show an analysis of resulting monthly payments for a range of interest rates and/or loan lengths near those provided by the operator. For example, if an interest rate of 9.5 percent were entered, display the monthly payments for 8.5, 9.0, 9.5, 10.0, and 10.5 percent.

MILEAGE

PURPOSE

For many of us, automobile operating efficiency is a continual concern. This program can help by keeping track of gasoline consumption, miles driven, and fuel milage for a motor vehicle. It allows reading and writing data files with the cassette unit. Thus, a master data file may be retained and updated. The program computes mileage (miles per gallon or MPG) obtained after each gasoline fill-up. A running log of all information is maintained. This enables trends in vehicle operating efficiency to be easily checked.

HOW TO USE IT

The program requests the following data from the operator as a record of each gasoline fill-up: date, odometer reading, and number of gallons purchased. The most useful results will be obtained if entries are chronological and complete, with each entry representing a full gasoline fill-up.

In order to use the cassette features, the operator must be able to position the tape correctly for both reading and writing. The simplest way to do this is to only record files at the beginning of a tape. One tape could certainly be used this way, with each file writing over the previous one. However, we suggest alternating between two physical tapes. This will insure a reasonably up-to-date backup tape in case of any failure.

The program operates from a central command mode. The operator requests branching to any one of five available routines.

When a routine completes execution, control returns to the command mode for any additional requests. A brief description of each routine follows:

1) READ OLD TAPE FILE

This reads previously stored data from the cassette. Any data already in memory is deleted.

2) KEY IN NEW DATA

This allows data records to be entered directly from the terminal. This mode is used to provide additional information after a cassette read and to enter data for the first time. The program will prompt the operator for the required information and then let him verify that it was entered correctly. A response of F to the verification request signals that no more data is to be entered.

3) CREATE NEW TAPE FILE

This command causes the current data to be written on cassette.

4) DISPLAY MILEAGE DATA

This routine computes mileage (miles per gallon) from the available data, and displays it in tabular form. Numbers are rounded to the nearest tenth. When data fills the screen, the user is prompted to press the **RETURN** key to continue the listing. When all data is displayed, pressing the **RETURN** key will reenter command mode.

5) END PROGRAM

This ends execution and returns the computer to BASIC.

SAMPLE RUN

```
MILEAGE CALCULATOR

User Options:

   Read old tape file -- enter '1'
   Key in new data -- enter '2'
   Create new tape file -- enter '3'
   Display mileage data -- enter '4'
   End program -- enter '5'
Your choice? 2
```

```
Data entry--please input:
   Date--month (numeric)? 9
       --day (numeric)? 27
       --year (nn or nnnn)? 79
   Odometer reading (miles)? 49Ø7.2
   Gas for fill-up (gallons)? 14.5

Is this correct?
   Date: 9 - 27 - 79
   Odometer: 49Ø7.2
   Gallons: 14.5
Enter 'Y' (yes), 'N' (no), or 'F'
   (yes, and finished)? Y

Data entry--please input:
                :
           (6 more entries are input)
                :
Enter 'Y' (yes), 'N' (no), or 'F'
   (yes, and finished)? F
                :
           (options menu listed again)
                :
Your choice? 3

To write a master file on unit #1--
   (1) Position new tape for writing
   (2) Press the RECORD button(s)
   (3) Hit the  RETURN  key
?_        (Recorder started, RETURN hit)
Recording . . .
Recording complete--turn off recorder
                :
           (options menu listed again)
                :
Your choice? 4

Date          Odometer      Gallons      MPG
9- 27- 79     4907.2        14.5         Ø
9- 30- 79     5212.8        12.7         24.1
1Ø- 5- 79     5487.5        14.4         19.1
1Ø- 8- 79     58Ø8.1        16.1         19.9
1Ø- 14- 79 6Ø94.4           14.8         19.3
1Ø- 23- 79 6418.6           15.7         2Ø.6
1Ø- 29- 79 6693.2           14.1         19.5

User Options:
      :
      :
Your choice? 5
```

PROGRAM LISTING

```
100 REM <MILES>
110 REM Mileage calculator / recorder
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 PRINT CHR$(12); "MILEAGE CALCULATOR"
160 PRINT: PRINT: CLEAR: MREC = 75: NREC = 0
170 DIM INFO(3, MREC)
200 PRINT "User Options:"
210 PRINT "   Read old tape file -- enter '1'"
220 PRINT "   Key in new data -- enter '2'"
230 PRINT "   Create new tape file -- enter '3'"
240 PRINT "   Display mileage data -- enter '4'"
250 PRINT "   End program -- enter '5'"
260 INPUT "Your choice"; R: PRINT: PRINT
270 ON R GOTO 400, 600, 500, 900, 290
280 PRINT "Try again!": GOTO 200
290 END
400 PRINT "To read a master file from unit #1--"
410 PRINT "   (1) Position the cassette for input"
420 PRINT "   (2) Press the cassette PLAY button"
430 PRINT "   (3) Hit the 'RETURN' key"
440 R$ = "": INPUT R$: PRINT "Loading . . . "
450 CLOAD*1 INFO
460 PRINT "Load complete--turn off the recorder."
470 NREC = NREC + 2: PRINT: GOTO 200
500 IF NREC < 1 THEN 910
505 PRINT "To write a master file on unit #1--"
510 PRINT "   (1) Position new tape for writing"
520 PRINT "   (2) Press the RECORD button(s)"
530 PRINT "   (3) Hit the 'RETURN' key"
540 R$ = "": INPUT R$: PRINT "Recording . . . "
545 FOR J = 1 TO 2000: NEXT J
550 CSAVE*1 INFO
560 PRINT "Recording complete--turn off recorder"
570 PRINT: GOTO 200
600 LI = 0: FOR J = MREC TO 1 STEP -1
610 NREC = J: IF INFO(2,J) <> 0 THEN 620
615 NEXT J
620 IF NREC < MREC THEN 660
630 PRINT "Sorry, array is now full.  Record all"
640 PRINT "   present data, and then restart."
650 GOTO 200
```

```
660 PRINT CHR$(12); "Data entry--please input:"
670 INPUT "   Date--month (numeric)"; M
673 INPUT "        --day (numeric)"; D
676 INPUT "        --year (nn or nnnn)"; Y
680 INPUT "   Odometer reading (miles)"; MILES
690 INPUT "   Gas for fill-up (gallons)"; GAL
695 IF Y > 100 THEN Y = Y - INT(Y/100) * 100
700 PRINT: PRINT "Is this correct?"
710 PRINT "   Date:"; M; "-"; D; "-"; Y
720 PRINT "   Odometer:"; MILES
730 PRINT "   Gallons:"; GAL
740 PRINT "Enter 'Y' (yes), 'N' (no), or 'F' ";
750 INPUT "(yes, and finished)"; W$
760 IF W$ = "N" OR W$ = "n" THEN 795
770 IF W$ = "F" OR W$ = "f" THEN LI = 1: GOTO 800
780 IF W$ = "Y" OR W$ ="y" THEN 800
790 PRINT: PRINT "Huh?": GOTO 700
795 PRINT "Please re-enter the data:": GOTO 670
800 IF INFO(2,NREC) <> 0 THEN NREC = NREC + 1
805 IF NREC > MREC THEN 630
810 INFO(1,NREC) = M * 10000 + D * 100 + Y
820 INFO(2,NREC) = MILES: INFO(3,NREC) = GAL
825 NREC = NREC + 1
830 IF LI = 1 THEN PRINT: PRINT: GOTO 200
840 GOTO 620
900 IF NREC > 1 THEN 920
910 PRINT "No data yet!": PRINT: GOTO 200
920 K = 1
930 PRINT CHR$(12); " Date"; TAB(13); "Odometer";
940 PRINT TAB(27); "Gallons"; TAB(41); "MPG"
950 FOR J=K TO K+16: IF INFO(2,J) = 0 THEN 1080
955 M = INT(INFO(1,J) / 10000)
960 D = INT((INFO(1,J) - M*10000) / 100)
970 Y = INFO(1,J) - M * 10000 - D * 100
980 IF J = 1 THEN MPG = 0: GOTO 1000
990 MPG = (INFO(2,J) - INFO(2,J-1)) / INFO(3,J)
1000 PRINT M; CHR$(8); "-"; D; CHR$(8); "-"; Y;
1010 PRINT TAB(14); INT(INFO(2,J)*10+0.5)/10;
1020 PRINT TAB(28); INT(INFO(3,J)*10+0.5)/10;
1030 PRINT TAB(42); INT(MPG * 10 + 0.5)/10
1040 IF J = MREC THEN 1080
1050 NEXT J: PRINT: PRINT: R$ = ""
1060 INPUT "Hit 'RETURN' to continue"; R$
1070 K = K + 16: GOTO 930
1080 PRINT: PRINT: R$ = ""
```

```
1090 INPUT "Hit 'RETURN' for options menu"; R$
1100 PRINT: PRINT: GOTO 200
```

EASY CHANGES

1. Changing the value of MREC in line 160 alters the maximum number of data records that the program allows. You may need to make MREC larger to accommodate additional data.
2. If you do not care about seeing the dates, they can be removed easily. This saves a little typing on data entry. To remove this feature, delete lines 670-676 and 710 entirely and change line 1000 to read

   1000 PRINT“----------”;

MAIN ROUTINES

160-170	Dimensioning and variable initialization.
200-290	Command mode. Displays available routines and branches to the operator’s choice.
400-470	Reads data from the cassette unit.
500-570	Writes data to the cassette unit after a delay loop.
600-840	Accepts terminal input.
900-1100	Calculates mileage and displays all information.

MAIN VARIABLES

MPG	Miles per gallon.
MREC	Maximum number of data records in memory.
NREC	Current number of data records in memory.
R	Command mode input.
W$,R$	Temporary string variables.
INFO	Data array, size 3*MREC, with coded dates in row 1, odometer readings in row 2, and fill-up amounts in row 3.
M,D,Y	Month, day, year.
J,K	Work and loop variables.
LI	Last input flag.

SUGGESTED PROJECTS

1. Calculate and print the average MPG over the whole data file.

This calculation can be done at the end of the DISPLAY MILEAGE routine between lines 1080 and 1090.

2. Add an option to do statistical calculations over a given subset of the data. The operator inputs a beginning and ending date. He is then shown things like average MPG, total miles driven, total gallons purchased, etc.; all computed only over the range requested.
3. Write a subroutine to graphically display MPG. A bar graph might work well.
4. Add a new parameter in each data record – the cost of each fill-up. Then compute things like the total cost of gasoline, miles/dollar, and state gasoline and sales tax totals.
5. Modify the program to accept data and print results using either English or metric units.

QUEST/EXAM

PURPOSE

Analyzing the results of a questionnaire or grading a multiple-choice examination can be a tedious and time-consuming process. This is particularly true if statistics for each question must be compiled. Since the basic idea behind computers is to allow programmers to be lazy, let's have the Sorcerer take most of the effort out of the task.

HOW TO USE IT

Make sure that the **SHIFT LOCK** key is down!

First the program asks if this is an exam (with answers to be checked against a key) or a questionnaire (with no "right" answers), and how many questions are to be scored. Up to fifty questions per exam or questionnaire can be used.

The next input specifies the maximum number of individual exams or questionnaires to be checked. This maximum is only used to dimension arrays, so fewer answer sets can always be used (memory size permitting).

An answer set can be referred to either by name or by number. Names should be unique—if there are two "SMITHs", add some initials to differentiate them. The answers themselves may be either numeric (1-9) or alphabetic (A-Z). The user inputs the largest possible "correct" value. For example, if each question had no more than 5 possible answer choices, the input value would be either **5** or **E**. If some wiseacre writes in "F-none of the above," it's still OK; the machine simply counts it as a wrong answer.

If this is an exam, the program next asks for the answer key. Input the answers as a single string of numbers or letters, with a length equal to the number of questions. The screen display includes an alignment guide to make input easier. All answers in the key—unlike the individual responses from each exam—must be less than or equal to the largest possible "correct" value.

Enter the answer sets one-by-one, as strings of letters or numbers. Again, an alignment guide is provided to help with input. If names are used, the program will ask for one before each answer set. Don't use a comma in the name unless it is entirely enclosed in quotation marks.

If a question was left blank, use a zero or any other nonblank character (other than one of the "legal" responses used in the answer key) in that location in the string. If a too-short or too-long string is input, the program will catch it and ask for a reentry.

When all individual answer sets have been entered, hit **RETURN** when asked for an answer string or name. This will call up the analysis options menu.

The first option checks all answers given to a single, specific question. A list of all possible legal answers, plus a category labeled "Other" (for blanks, and that wiseacre noted above), is produced. This list shows how many persons gave each response.

The second option is similar, but it analyzes all questions in order. The routine waits for the user to hit **RETURN** after analysis of each single question, so you will have time to note the answers.

The third option shows the answers from a given exam or questionnaire, specified by name (if selected) or number. If this is an exam, the answer key is also shown for comparison.

The fourth option is similar, but shows the responses from each individual answer set, in order. The routine pauses after the screen fills, until **RETURN** is hit.

Option five allows more data to be added and, in turn, analyzed. Options six and seven restart and stop the program, respectively.

SAMPLE RUN

```
QUESTIONNAIRE/EXAM ANALYSIS ROUTINE

Be sure 'SHIFT LOCK' is down!!

Is this a questionnaire (enter 'Q') or an
exam (enter 'E')? E

How many questions (1 -- 50)? 22

Maximum number of answer sets? 5Ø

Are answer sets named (enter 'Y' or 'N')? Y

Are answers numbers (enter 'N') or letters
(enter 'L')? L

What is the largest letter (B -- Z) used as
an answer? E

Input the exam's answer key now --
          1         2
 123456789Ø123456789Ø12
? ADCBAEDBAECCEBABBEDDAB

Entry # 1 -- name? TOM
Enter answers (or 'RETURN') --
          1         2
 123456789Ø123456789Ø12
? ADCBADEBAECCDBBCBEDDAE
16 correct ( 72.7273 %)

Entry # 2 -- name? JIM
Enter answers (or 'RETURN') --
          1         2
 123456789Ø123456789Ø12
? ADCBAEEEAECCEBCBCEDDAA
    :
(later . . .)
    :
Entry # 8 -- name? X
Enter answers (or 'RETURN') --
          1         2
 123456789Ø123456789Ø12
? _ (RETURN hit)

Average score was 42.8751 % over 7 entries.

Options menu:
   Analyze one question -- enter '1'
   Analyze all questions -- enter '2'
   Review one entry -- enter '3'
   Review all entries -- enter '4
   Add more entries -- enter '5'
   Start over -- enter '6'
   End program -- enter '7'
Your choice (1 -- 7)? 1

What question number? 12

Question # 12
Key answer is C.

Response  Frequency  Percent
A         Ø          Ø
B         2          28.5714
C         3          42.8571
D         2          28.5714
E         Ø          Ø
Other     Ø          Ø

Options menu:
  :
(Options listed again)
  :
Your choice (1 -- 7)? 3

Entry name? LIZ

Answer key --

           1         2
  12345678901234567890Ø12

Entry # 5 -- LIZ
           1         2
  1234567890123456789Ø12
  ADDBAEDBAEECEAABAEADDB

Options menu:
  :
(Options listed again)
  :
Your choice (1 -- 7)? 7
```

PROGRAM LISTING

```
1ØØ REM <QEXAM>
11Ø REM Questionnaire / exam analysis
12Ø REM For the Sorcerer (TM Exidy Inc.)
13Ø REM Copyright 1979
```

```
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 CLEAR 3000: E$ = "ERROR -- try it again!"
160 PR$ = "Press 'RETURN' to continue"
170 HD$ = "QUESTIONNAIRE / EXAM ANALYSIS ROUTINE"
180 PRINT CHR$(12); HD$: PRINT: PRINT
190 PRINT "Be sure 'SHIFT LOCK' is down!!": PRINT
200 PRINT "Is this a questionnaire (enter 'Q') ";
210 INPUT "or an exam (enter 'E')"; R$: TFLAG = 0
220 PRINT: IF R$ = "Q" THEN 240
230 TFLAG = 1: IF R$ <> "E" THEN PRINT E$:GOTO 200
240 INPUT "How many questions (1 -- 50)"; R$
250 NQ = INT(VAL(R$)): PRINT
260 IF NQ < 1 OR NQ > 50 THEN PRINT E$: GOTO 240
270 INPUT "Maximum number of answer sets"; R$
280 ME = INT(VAL(R$)): PRINT
290 IF ME < 1 THEN PRINT E$: GOTO 270
295 DIM EN$(ME)
300 PRINT "Are answer sets named (enter 'Y' or ";
310 INPUT "'N')"; R$: NFLAG = 0: PRINT
320 IF R$ = "N" THEN 350
330 NFLAG=1: IF R$ <> "Y" THEN PRINT E$: GOTO 300
340 DIM NAME$(ME)
350 PRINT "Are answers numbers (enter 'N') or ";
360 INPUT "letters (enter 'L')"; R$
370 PRINT: IF R$ = "L" THEN 420
380 IF R$ <> "N" THEN PRINT E$: GOTO 350
390 INPUT "Choices per question (2 -- 9)"; NC
400 PRINT: IF NC<2 OR NC>9 THEN PRINT E$: GOTO 390
410 MIN = 49: MAX = 48 + NC: GOTO 500
420 PRINT "What is the largest letter (B -- Z) ";
430 INPUT "used as an answer"; R$: PRINT
440 MIN = 65: MAX = ASC(R$): NC = MAX - 64
450 IF NC < 2 OR NC > 26 THEN PRINT E$: GOTO 420
500 DIM TALLY(NC): PRINT CHR$(12); HD$: PRINT
510 AK$ = "": IF TFLAG = 0 THEN 600
520 PRINT "Input the exam's answer key now --"
530 GOSUB 900: INPUT AK$: PRINT: GOSUB 950
540 IF W = 1 THEN PRINT E$: GOTO 520
600 QEN = 1: TR = 0
610 RA = 0: PRINT: PRINT: PRINT "Entry #"; QEN;
620 IF NFLAG = 1 THEN INPUT "-- name"; NAME$(QEN)
630 PRINT: PRINT "Enter answers (or 'RETURN') --"
640 GOSUB 900: INPUT EN$(QEN): PRINT
650 IF EN$(QEN) = "" THEN 720
655 IF LEN(EN$(QEN)) <> NQ THEN PRINT E$: GOTO 630
```

```
660 IF TFLAG = 0 THEN 710
670 FOR J = 1 TO NQ: W$ = MID$(AK$,J,1)
680 IF W$ = MID$(EN$(QEN),J,1) THEN RA = RA + 1
690 NEXT J: TR = TR + RA: PRINT RA;
700 PRINT "correct (": RA*100/NQ; "%)"
710 QEN = QEN + 1: IF QEN <= NE THEN 610
720 NE = QEN - 1: PRINT: IF TFLAG = 0 THEN 800
730 PRINT "Average score was";
740 PRINT TR*100/(NQ*NE); "% over"; NE; "entries."
800 PRINT: PRINT: PRINT "Options menu:"
805 PRINT "   Analyze one question -- enter '1'"
810 PRINT "   Analyze all questions -- enter '2'"
815 PRINT "   Review one entry -- enter '3'"
820 PRINT "   Review all entries -- enter '4'"
825 PRINT "   Add more entries -- enter '5'"
830 PRINT "   Start over -- enter '6'"
835 PRINT "   End program -- enter '7'"
840 INPUT "Your choice (1 -- 7)"; R$
850 R = INT(VAL(R$))
860 IF R < 1 OR R > 7 THEN PRINT E$: GOTO 800
870 ON R GOTO 1000,2000,3000,4000,5000,6000: END
900 W = INT(NQ / 10): IF W < 1 THEN 920
910 FOR K=1 TO W: PRINT TAB(K*10);K;: NEXT K: PRINT
920 PRINT TAB(2);: FOR K = 1 TO NQ
930 W$ = RIGHT$(STR$(K),1): PRINT W$;:NEXT K: PRINT
940 RETURN
950 W = 0: IF LEN(AK$) <> NQ THEN W = 1: RETURN
960 FOR K = 1 TO NQ: W$ = MID$(AK$,K,1)
970 IF ASC(W$)<MIN OR ASC(W$)>MAX THEN W=1: RETURN
980 NEXT K: RETURN
1000 PRINT CHR$(12); HD$: PRINT: PRINT
1010 INPUT "What question number"; R$
1020 PRINT: QN = INT(VAL(R$))
1030 IF QN < 1 OR QN > NQ THEN PRINT E$: GOTO 1010
1040 GOSUB 1500: GOTO 800
1500 PRINT: PRINT "Question #"; QN
1505 IF TFLAG = 0 THEN 1520
1510 PRINT "Key answer is "; MID$(AK$,QN,1); "."
1520 PRINT: PRINT "Response","Frequency","Percent"
1525 FOR K = 0 TO NC: TALLY(K) = 0: NEXT K
1530 FOR K = 1 TO NE: W=ASC(MID$(EN$(K),QN,1))
1540 IF W < MIN OR W > MAX THEN 1560
1550 W = W-MIN+1: TALLY(W) = TALLY(W) + 1:GOTO 1570
1560 TALLY(0) = TALLY(0) + 1
1570 NEXT K: FOR K = 1 TO NC: PRINT CHR$(MIN+K-1),
1580 PRINT TALLY(K), TALLY(K)*100/NE: NEXT K
1590 PRINT "Other", TALLY(0), TALLY(0)*100/NE
1600 RETURN
2000 PRINT CHR$(12); HD$: PRINT: PRINT
2010 FOR QN = 1 TO NQ: GOSUB 1500: PRINT:PRINT PR$;
2020 INPUT R$: NEXT QN: GOTO 800
3000 PRINT CHR$(12);HD$:PRINT:IF NFLAG=1 THEN 3040
3010 INPUT "Entry number"; R$: QEN = INT(VAL(R$))
3020 IF QEN < 1 OR QEN > NE THEN PRINT E$:GOTO 3010
3030 GOTO 3070
3040 INPUT "Entry name"; R$: FOR J = 1 TO NE
3050 IF NAME$(J) = R$ THEN QEN = J: GOTO 3070
3060 NEXT J: PRINT "Not found!": GOTO 3040
3070 IF TFLAG = 0 THEN 3100
3080 PRINT: PRINT "Answer key --": GOSUB 900
3090 PRINT TAB(2); AK$
3100 PRINT: PRINT "Entry #"; QEN; "-- ";
3110 IF NFLAG = 1 THEN PRINT NAME$(QEN);
3120 PRINT: GOSUB 900: PRINT TAB(2); EN$(QEN)
3130 GOTO 800
4000 J = 1
4010 PRINT CHR$(12); HD$: PRINT: PRINT
4020 IF TFLAG = 0 THEN 4050
4030 PRINT "Answer key --": GOSUB 900
4040 PRINT TAB(2); AK$: PRINT
4050 PRINT "Entry #"; J; "-- ";
4060 IF NFLAG = 1 THEN PRINT NAME$(J);
4070 PRINT:GOSUB 900:PRINT TAB(2);EN$(J):PRINT
4080 J = J + 1: IF J > NE THEN 4100
4090 IF J <> INT(J/5) * 5 THEN 4050
4100 PRINT PR$;: INPUT R$: IF J <= NE THEN 4010
4110 GOTO 800
5000 QEN = NE: PRINT CHR$(12): GOTO 710
6000 GOTO 150
```

EASY CHANGES

1. Up to 60 questions per questionnaire or exam could be used with the present display routine. Change the value of 50 in lines 240 and 260 to the desired value.
2. If a ?/**OS** error occurs, string space is too small. Change the value of 3000 in line 150 to a larger value, in order to accommodate long or numerous names and answer strings.

MAIN ROUTINES

150-190	Initializes variables, prints headings.
200-510	Initialization dialog. Selects options, dimensions arrays.
520-540	Gets and checks answer key, if any.
600-610	Initializes input counter and score accumulators. Prints headings for entry number QEN.
620	Gets name for QEN answer set, if any.
630-660	Gets QEN answer set.
670-700	Scores QEN answer set, if applicable.
710-740	Displays average score, if applicable.
800-870	Option menu. Gets user's choice, branches to appropriate routine.
900-940	Displays alignment guide numbers.
950-980	Checks answer key for proper length and contents.
1000-1600	Analyzes answers to a single question.
2000-2020	Analyzes all questions.
3000-3130	Displays a single answer set.
4000-4110	Displays all answer sets.
5000	Resets QEN for additional entries.
6000	Restarts program.

MAIN VARIABLES

E$,PR$,HD$	Output message strings.
W$,W	Work variables.
R$,R	User responses.
NQ	Number of questions.
NE	Number of answer sets input.
ME	Maximum number of answer sets.
NC	Number of choices per question.
MIN,MAX	Minimum and maximum ASCII values for "legal" answers.
RA	Number of right answers in one set.
TR	Total number of right answers for all sets.
J,K,L	Loop counters.
QN	Question number.
QEN	Questionnaire/exam number.
TFLAG	Type flag (0 = questionnaire, 1 = exam).
NFLAG	Name flag (0 = none, 1 = named).

LFLAG	Answer flag (0 = letters, 1 = numbers).
EN$	Array of answer strings.
AK$	Answer key string.
NAME$	Array of respondents' names.
TALLY	Array used to tally answer frequencies.

SUGGESTED PROJECTS

1. Add an option to change the answer key after the data for the exams is entered. This would be useful in case a mistake was found when reviewing the data.
2. Add an option to allow the operator to rescore each of the exams after all are entered.
3. Combine the functions of the STATS program with QEXAM.

Section 2
Educational Programs

INTRODUCTION TO EDUCATION PROGRAMS

Education is one area when computers are certain to have more and more impact. Though a computer cannot completely replace a human teacher, the machine does have certain advantages. It is ready any time you are, allows you to go at your own pace, handles rote drill effortlessly, and is devoid of any personality conflicts.

With a good software library, the Sorcerer can be a valuable learning center in the school or at home. Here are six programs to get you started.

Mathematics is certainly a "natural" subject for computers. NUMBR is designed for preschool children. While familiarizing youngsters with computers, it provides an entertaining way for them to learn numbers and elementary counting. MATH is aimed at older, grade-school students. It provides drill in various kinds of math problems. The child can adjust the difficulty factors, allowing the program to be useful for several years.

By no means is the Sorcerer restricted to mathematical disciplines. We include two programs designed to improve your word skills. VOCAB will help you expand your vocabulary. TACHI turns the Sorcerer into a reading clinic, helping you to improve your reading speed.

Do you have trouble familiarizing yourself with the increasingly prevalent metric system? METRC is the answer.

Need help learning a certain subject? FLASH allows you to create your own "computer flashcards." Then you can drill yourself until you get it right.

MATH

PURPOSE

MATH provides mathematics drills for grade-school children. The student can request problems in addition, subtraction, or multiplication from the program. Also, he or she may ask that the problems be easy, medium, or hard. The program should be useful to a child over an extended period of time. He can progress naturally to a harder category of problems when he begins to regularly perform well at one level. The difficulty and types of problems encompass those normally encountered by school children between the ages of six and ten.

The problems are constructed randomly within the constraints imposed by the degree of difficulty selected. This gives the student fresh practice each time the program is used. After entering answers, he is told whether he was right or wrong. The correct answers are also displayed.

HOW TO USE IT

To begin, the student must indicate what type of problem he wishes to do. The program requests an input of 1, 2, or 3 to indicate addition, subtraction, or multiplication, respectively. It then asks whether easy, medium, or hard problems are desired. Again an input of 1, 2, or 3 is required. Simply hit the key— **RETURN** isn't needed for these initial entries.

The screen will clear and five problems of the desired type will be displayed. The user now begins to enter his answers to each problem.

A question mark is used to prompt the user for each digit of the answer, one digit at a time. This is done moving right to left, the way arithmetic problems are naturally solved.

To start each problem, the question mark will appear in the spot for the rightmost (or units column) digit of the answer. When the key for a digit from 0-9 is pressed, that digit will replace the question mark on the screen. The question mark moves to the immediate left waiting for a digit for the "tens" column.

Digits are entered in this right to left manner until the complete answer has been input. Then the **RETURN** key must be pressed. This will end the answer to the current problem and move the question mark to begin the answer for the next question.

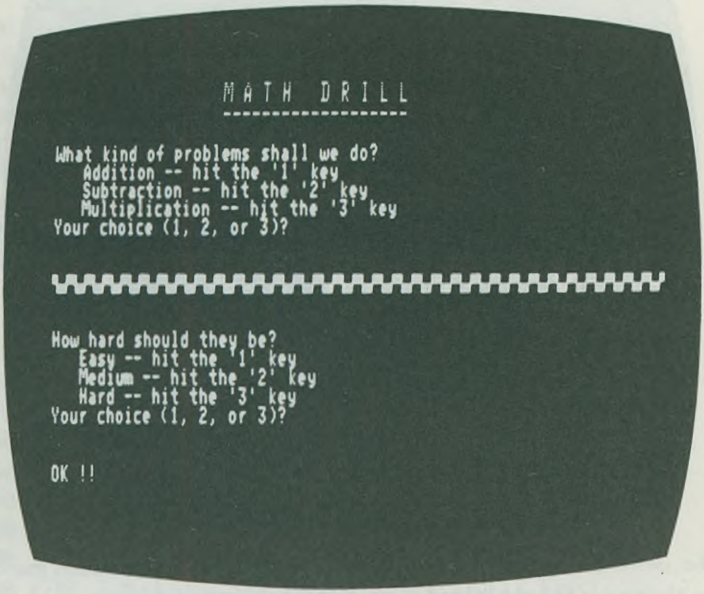
If the **RETURN** key is pressed to begin a problem, an answer of zero is assumed to be intended. No problems created by this program have answers of more than three digits. If a 4-digit answer is given, the program will accept the answer, but then go immediately to the next problem. Answers to the problems are never negative.

The program will display the correct answers to the five problems on the screen after the student has entered his five answers. The message "Right!" or "Wrong!" will also be displayed below each problem.

Then the message "Hit any key to continue" will be displayed. After a key is pressed, a new set of five problems of the same type will be presented.

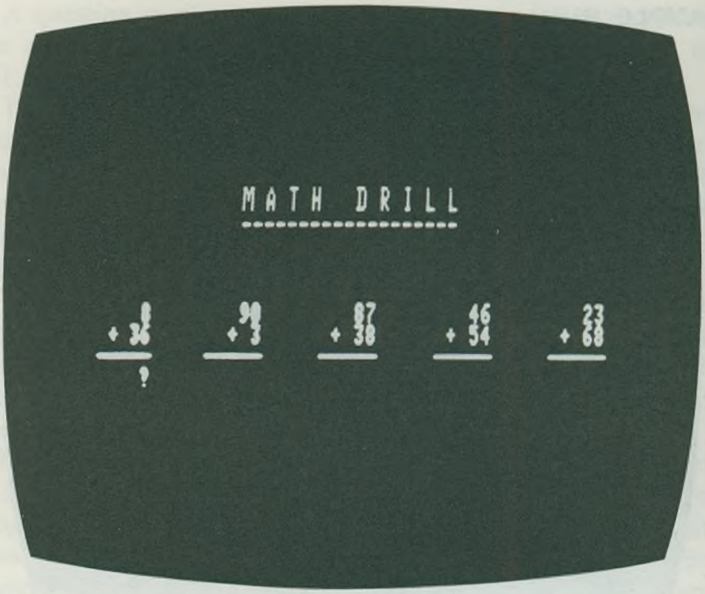
This continues until twenty problems have been worked. Before ending, the program shows what the student's performance has been. This is expressed as the number of problems solved correctly and also as the percentage of problems solved correctly.

SAMPLE RUN



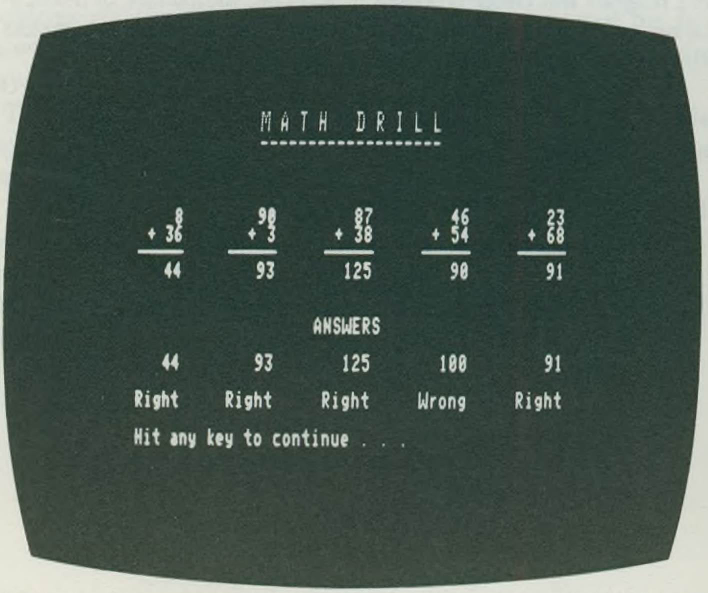
```
             M A T H   D R I L L
             -------------------

What kind of problems shall we do?
   Addition -- hit the '1' key
   Subtraction -- hit the '2' key
   Multiplication -- hit the '3' key
Your choice (1, 2, or 3)?


How hard should they be?
   Easy -- hit the '1' key
   Medium -- hit the '2' key
   Hard -- hit the '3' key
Your choice (1, 2, or 3)?

OK !!
```

The program will create problems in addition, subtraction, or multiplication, in any of three ranges of difficulty. The user makes his choices simply by the **1**, **2**, or **3** keys, without using **RETURN**.



```
                MATH DRILL
                ----------

       8       90       87       46       23
    + 36     +  3     + 38     + 54     + 68
    ----     ----     ----     ----     ----
       ?
```

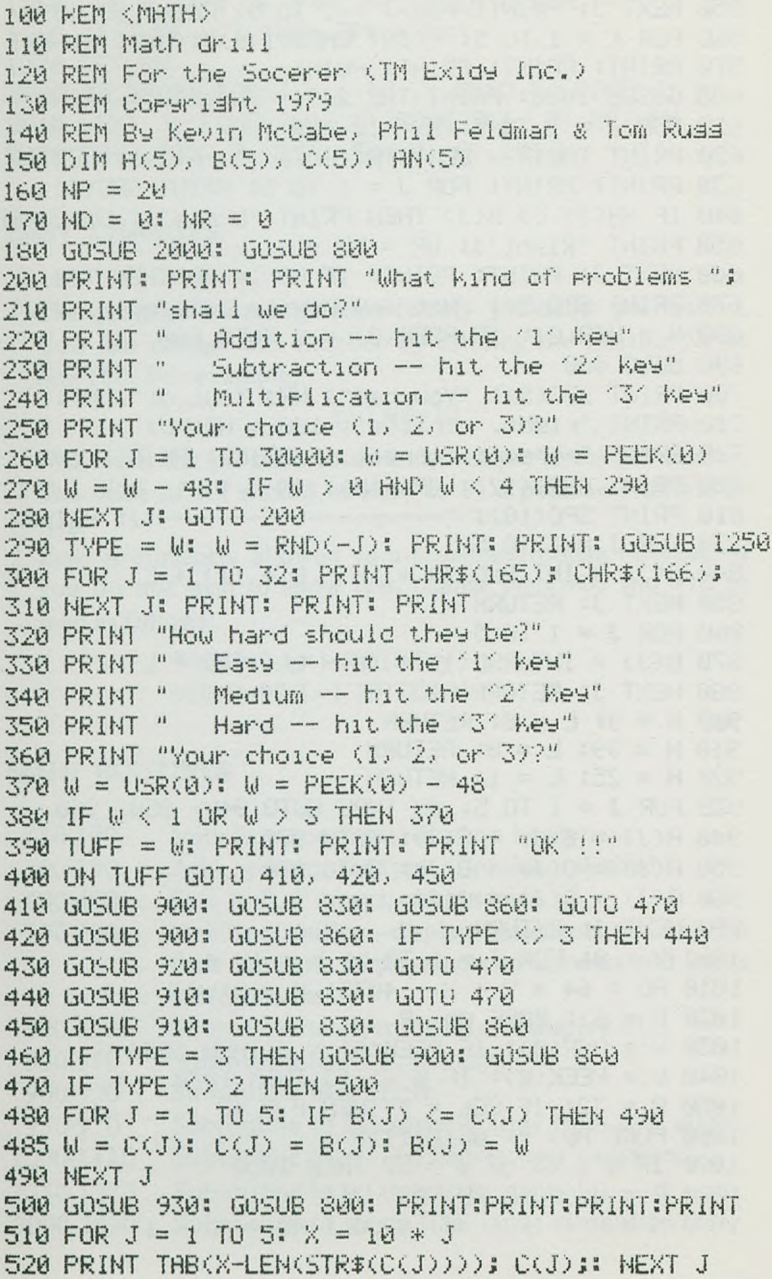
An initial set of five hard addition problems is shown. A question mark in the proper column is used to prompt the operator for input.



```
                MATH DRILL
                ----------

       8       90       87       46       23
    + 36     +  3     + 38     + 54     + 68
    ----     ----     ----     ----     ----
      44       93      125       90       91

                 ANSWERS

      44       93      125      100       91
   Right    Right    Right    Wrong    Right
   Hit any key to continue . . .
```

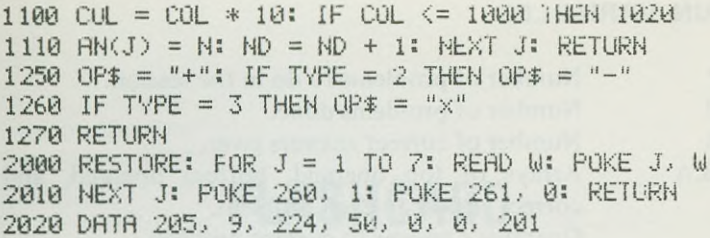
The operator has entered five answers. The correct answers, with appropriate "Right" or "Wrong" comments, are displayed. The next set of five problems will be displayed whenever any key is pressed.

PROGRAM LISTING



```
100 REM <MATH>
110 REM Math drill
120 REM For the Socerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 DIM A(5), B(5), C(5), AN(5)
160 NP = 20
170 ND = 0: NR = 0
180 GOSUB 2000: GOSUB 800
200 PRINT: PRINT: PRINT "What kind of problems ";
210 PRINT "shall we do?"
220 PRINT "   Addition -- hit the '1' key"
230 PRINT "   Subtraction -- hit the '2' key"
240 PRINT "   Multiplication -- hit the '3' key"
250 PRINT "Your choice (1, 2, or 3)?"
260 FOR J = 1 TO 30000: W = USR(0): W = PEEK(0)
270 W = W - 48: IF W > 0 AND W < 4 THEN 290
280 NEXT J: GOTO 200
290 TYPE = W: W = RND(-J): PRINT: PRINT: GOSUB 1250
300 FOR J = 1 TO 32: PRINT CHR$(165); CHR$(166);
310 NEXT J: PRINT: PRINT: PRINT
320 PRINT "How hard should they be?"
330 PRINT "   Easy -- hit the '1' key"
340 PRINT "   Medium -- hit the '2' key"
350 PRINT "   Hard -- hit the '3' key"
360 PRINT "Your choice (1, 2, or 3)?"
370 W = USR(0): W = PEEK(0) - 48
380 IF W < 1 OR W > 3 THEN 370
390 TUFF = W: PRINT: PRINT: PRINT "OK !!"
400 ON TUFF GOTO 410, 420, 450
410 GOSUB 900: GOSUB 830: GOSUB 860: GOTO 470
420 GOSUB 900: GOSUB 860: IF TYPE <> 3 THEN 440
430 GOSUB 920: GOSUB 830: GOTO 470
440 GOSUB 910: GOSUB 830: GOTO 470
450 GOSUB 910: GOSUB 830: GOSUB 860
460 IF TYPE = 3 THEN GOSUB 900: GOSUB 860
470 IF TYPE <> 2 THEN 500
480 FOR J = 1 TO 5: IF B(J) <= C(J) THEN 490
485 W = C(J): C(J) = B(J): B(J) = W
490 NEXT J
500 GOSUB 930: GOSUB 800: PRINT:PRINT:PRINT:PRINT
510 FOR J = 1 TO 5: X = 10 * J
520 PRINT TAB(X-LEN(STR$(C(J)))); C(J);: NEXT J
```

```
530 PRINT: FOR J = 1 TO 5: X = 10 * J
540 PRINT TAB(X - LEN(STR$(B(J))) - 1); OP$; B(J);
550 NEXT J: PRINT: FOR J = 1 TO 5: PRINT SPC(5);
560 FOR K = 1 TO 5: PRINT CHR$(140);: NEXT K:NEXT J
570 PRINT: PRINT: PRINT: PRINT
600 GOSUB 1000: PRINT TAB(24); "ANSWERS": PRINT
610 FOR J = 1 TO 5: X = 10 * J
620 PRINT TAB(X - LEN(STR$(A(J)))); A(J);: NEXT J
630 PRINT: PRINT: FOR J = 1 TO 5: PRINT SPC(5);
640 IF AN(J) <> A(J) THEN PRINT "Wrong";: GOTO 660
650 PRINT "Right";: NR = NR + 1
660 NEXT J: PRINT: PRINT: IF ND >= NP THEN 700
670 PRINT SPC(5); "Hit any key to continue . . ."
680 W = USR(0): IF PEEK(0) = 0 THEN 680
690 GOTO 400
700 PRINT SPC(5); "You had"; NR; "out of"; ND;
710 PRINT "right.  That's"; 100 * NR/ND;
720 PRINT "percent correct.": PRINT: PRINT: END
800 PRINT CHR$(12); SPC(18); "M A T H   D R I L L"
810 PRINT SPC(18); "-------------------";: RETURN
830 FOR J = 1 TO 5
840 C(J) = INT(RND(1) * (H - L + 1)) + L
850 NEXT J: RETURN
860 FOR J = 1 TO 5
870 B(J) = INT(RND(1) * (H - L + 1)) + L
880 NEXT J: RETURN
900 H = 9: L = 0: RETURN
910 H = 99: L = 0: RETURN
920 H = 25: L = 1: RETURN
930 FOR J = 1 TO 5: ON TYPE GOTO 940, 950, 960
940 A(J) = B(J) + C(J): GOTO 970
950 A(J) = C(J) - B(J): GOTO 970
960 A(J) = B(J) * C(J)
970 NEXT J: RETURN
1000 Y = 9: FOR J = 1 TO 5: X = 10 * J
1010 AD = 64 * Y + X - 4033: N = 0: COL = 1
1020 P = 63: POKE AD, P
1030 W = USR(0): IF PEEK(0) = 0 THEN 1030
1040 W = PEEK(0): IF W <> 13 THEN 1070
1050 P = 32: IF COL = 1 THEN P = 48
1060 POKE AD, P: GOTO 1110
1070 IF W < 48 OR W > 57 THEN 1030
1080 P = W: POKE AD, P
1090 N = N + (P - 48) * COL: AD = AD - 1
```



```
1100 COL = COL * 10: IF COL <= 1000 THEN 1020
1110 AN(J) = N: ND = ND + 1: NEXT J: RETURN
1250 OP$ = "+": IF TYPE = 2 THEN OP$ = "-"
1260 IF TYPE = 3 THEN OP$ = "x"
1270 RETURN
2000 RESTORE: FOR J = 1 TO 7: READ W: POKE J, W
2010 NEXT J: POKE 260, 1: POKE 261, 0: RETURN
2020 DATA 205, 9, 224, 50, 0, 0, 201
```

EASY CHANGES

1. The program currently does twenty problems per session. You can change this number by altering the variable NP in line 160. For example,

   160 NP=10

   will cause the program to do only ten problems per session. The value of NP should be kept to be a positive multiple of five.
2. Zero is currently allowed as a possible problem operand. If you do not wish to allow this, change lines 900 and 910 to read as follows:

   900 H=9:L=1:RETURN
   910 H=99:L=1:RETURN

MAIN ROUTINES

150-180	Initializes constants, displays header.
200-390	Asks operator for type of problem desired.
400-490	Sets A, B, C arrays, clears screen.
500-720	Mainline routine—displays problems, gets operator's answers, displays correct answers and user's performance.
800-810	Subroutine to clear screen and display title.
830-880	Subroutine to set B, C arrays.
900-920	Subroutine to set L, H.
930-970	Subroutine to calculate A array from B, C arrays.
1000-1110	Subroutine to get and display user's answers.
1250-1270	Subroutine to set operation symbol.
2000-2020	Subroutine to set up USR input subroutine.

MAIN VARIABLES

NP	Number of problems to do in the session.
ND	Number of problems done.
NR	Number of correct answers given.
C,B,A	Arrays of top operand, bottom operand, and correct answer to each problem.
N	Operator's answer to current problem.
AN	Array of operator's answers.
TYPE	Type of problems requested (1 = addition, 2 = subtraction, 3 = multiplication).
TUFF	Kind of problem requested (1 = easy, 2 = medium, 3 = hard).
H,L	Highest, lowest integers to allow as problem operands.
COL	Answer column being worked on.
X,Y	Horizontal, vertical screen position of cursor, measured from the top left corner.
W	Work variable.
J,K	Loop indices.
P	POKE value.

SUGGESTED PROJECTS

1. Keep track of problems missed and repeat them quickly for additional practice.
2. No negative operands or answers are currently allowed. Rewrite the problem generation routines and the operator's answer routines to allow the possibility of negative answers.
3. The answers are now restricted to 3-digit numbers. However, the program will workfine for 4-digit numbers if the operands of the problems are allowed to be large enough. Dig into the routines at lines 400-490, 900-920, and 1100. See how they work and then modify them to allow possible 4-digit answers.
4. The operator cannot currently correct any mistakes he makes while typing in his answers. Modify the program to allow him to do so.
5. Modify the program to allow problems in division.
6. How does the Sorcerer "know" that a key is pressed? Lines 2000-2020 set up a machine-language subroutine that can be called using the USR operation of BASIC. See the WALLS program for more details.

FLASHCARD

PURPOSE

There are certain things that the human mind is capable of learning only through repetition. Not many people can remember the multiplication tables after their first exposure, for example. The same applies to learning the vocabulary of a foreign language, the capital cities of the fifty states, or famous dates in history. The best way to learn them is to simply review them over and over until you have them memorized.

A common technique for doing this involves the use of flashcards. You write one half of the two related pieces of information on one side of a card, and the other half on the other side. After creating a set of these cards, you can drill yourself on them over and over until you always remember what's on the other side of the card.

But why waste precious natural resources by using cards? Use your computer instead. This program lets you create flashcards, drill using them, and save them on cassette tape for later review.

HOW TO USE IT

The Sorcerer is a great machine – except when it comes time to save alphabetic data on tape. Many machines will allow ASCII data to be stored on tape just like numeric arrays, without messy conversions or excessive memory requirements. The Sorcerer, though, makes life more difficult. Before a set of ASCII values can be stored on a cassette tape, each value must be stored in an array, in numeric rather than in character form.

This program is a trade-off, requiring slightly more work from the user in return for a much easier method of saving alphabetic data. The "flashcards"—really just pairs of strings—are contained in DATA statements. Each DATA statement must be added to the "blank" program presented here at lines 6000-9999. In order to use the same flashcards at a later date, the entire program is saved on tape using **CSAVE**, just like any other program.

The program has five menu-driven options. The first is the "drill" routine, the heart of the program. One side—that is, one of the two strings—of a flashcard is displayed on the screen. Both the card and the side are chosen at random. If you respond with the string corresponding to the correct string on the other side of the card, the machine says, "That's right!" Otherwise, the correct answer will be displayed. Entering ***MENU** returns you to the options menu, after a summary of the number of correct responses.

Options 2, 3, and 4 are used to add DATA statements and to store the routine. Note that these options simply print instructions. The actual addition of DATA strings or storing of the program on tape occurs *after* the routine ends, by explicit user input. Option five simply ends the routine, with a reminder to **CSAVE** the flashcards if you want to use the same set again in another session.

When saving flashcard sets on a tape with more than one set, be sure not to wipe out prior sets or the "blank" version of the program. As written, each version of the program must have between five and 25 pairs of flashcard strings. See "Easy Changes" to alter these limits. Note that, during the drill sequence, a card will not be repeated for a certain number of questions. This number is one less than the selected minimum number of cards. Any card used in the last four questions won't be used again.

SAMPLE RUN

```
**** FLASHCARDS ****

Options menu:
   Drill on current set -- enter '1'
   Create a new set -- enter '2'
   Add to current set -- enter '3'
   Store current set -- enter '4'
   End program -- enter '5'
Your choice? 2


New flashcards -- option # 2

Both sides of each flashcard are stored in DATA
statements.  The two sides are to be put in
either one DATA statement, or in two successive
DATA statements.  All items in the DATA
statements must be enclosed in quotation marks.

Be sure that all entries USE UPPERCASE letters!

Number the data statements between 6000 and
9999, in order. After you've entered the new
DATA, input 'RUN' to either use the new set, or
store it on tape.

To double-check that no DATA statements are now
used, a list will be printed.  When you're
ready, hit 'RETURN'?_    (RETURN hit)

5999 END: REM -- put DATA in lines 6000-9999
10000 DATA "XXX","XXX": REM -- stop flag
10001 REM
10002 REM REMEMBER--  1) Proper DATA format
10003 REM             2) Proper DATA line numbers
10004 REM             3) Use 'CSAVE' to save
10005 REM             4) Use 'RUN' to start over
READY

6000 DATA "GROSS WEIGHT", "1670 LBS"
6010 DATA "RANGE", "690 MILES"
6020 DATA "FUEL CAPACITY", "37.5 GALLONS"
6030 DATA "MAX. BAGGAGE", "120 LBS"
6040 DATA "T/O ROLL", "1340 FT"
6050 DATA "STALL-CLEAN", "48 KNOTS"
6060 DATA "STALL-DIRTY", "43 KNOTS"
RUN


**** FLASHCARDS ****
    :
(Options menu again)
    :
 Your choice? 1


 Flashcard drill -- option #1

 I'll display one side of a flashcard.  Tell me
 what's on the other side, using UPPERCASE
 letters.
```

```
Side one -- RANGE
Side two (or *MENU) --? 690 MILES
That's right!

Side one -- 3 7.5 GALLONS
Side two (or *MENU) --? OIL CAPACITY
Sorry, it's FUEL CAPACITY.

Side one -- 1670 LBS
Side two (OR *MENU) --? GROSS WEIGHT
That's right!

Side one -- MAX. BAGGAGE
Side two (or *MENU) --? *MENU

You had 2 of 3 correct (66.6667 %).
  :
(Options menu again)
  :
Your choice? 5

**** CAUTION ****

Have you stored the current set yet? If you
forgot, enter '*MENU' and do it now. But if you
REALLY want to quit now, just hit 'RETURN'?__
(RETURN hit)
```

PROGRAM LISTING

```
100 REM <FLASH>
110 REM Flashcard creation / display
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 CLEAR 2000: MN = 5: MX = 25
160 HD$ = "**** FLASHCARDS ****"
170 CM$ = "**** CAUTION ****"
180 PRINT CHR$(12); HD$
190 DIM FC$(MX,2), PC(MN-1)
200 PRINT: PRINT: PRINT "Options menu:"
210 PRINT "   Drill on current set -- enter '1'"
220 PRINT "   Create a new set -- enter '2'"
230 PRINT "   Add to current set -- enter '3'"
240 PRINT "   Store current set -- enter '4'"
250 PRINT "   End program -- enter '5'"
260 INPUT "Your choice"; R$: R = INT(VAL(R$))
270 IF R<1 OR R>5 THEN PRINT "Say again?":GOTO 260
280 ON R GOSUB 1000,2000,3000,4000,5000: GOTO 200
```

```
1000 PRINT CHR$(12); HD$: PRINT
1010 PRINT "Flashcard drill -- option #1": PRINT
1020 PRINT: RESTORE: RA=0: WA=0: NCHRD=0: PRINT
1030 READ W$, X$
1040 IF X$ = W$ AND W$ = "XXX" THEN 1200
1050 IF X$ <> "XXX" AND W$ <> "XXX" THEN 1090
1060 PRINT "ERROR--the number of front and back ";
1070 PRINT "sides doesn´t match!"
1080 PRINT "RUN ABORTED!!  Check the listing!": END
1090 NCARD = NCARD + 1: FC$(NCARD,1) = W$
1100 FC$(NCARD,2) = X$: IF NCARD < 50 THEN 1030
1110 PRINT "Too many cards -- limit is"; MX: END
1200 IF NCARD<5 THEN PRINT "Too few cards!":RETURN
1205 PRINT "I´ll display one side of a flashcard."
1210 PRINT: PRINT "Tell me what´s on the other ";
1220 PRINT "side, using UPPERCASE letters."
1230 N = INT(RND(1)*NCARD + 0.99)
1235 IF N < 1 THEN 1230
1240 FOR J = 1 TO MN-1: IF N = PF(J) THEN 1230
1250 NEXT J: FOR J = MN - 2 TO 1 STEP -1
1260 PF(J+1) = PF(J): NEXT J: PF(1) = N
1270 S1=1: S2=2: IF RND(1) > .5 THEN S1=2: S2=1
1300 PRINT: PRINT
1310 PRINT "Side one -- "; FC$(N,S1)
1320 INPUT "Side two (or ´*MENU´) --"; R$
1330 IF R$ <> "*MENU" THEN 1370
1340 PRINT: PRINT: PRINT "You had"; RA; "of";
1350 PRINT RA + WA; "correct (";
1355 IF RA + WA = 0 THEN PRINT "0 %).": RETURN
1360 PRINT 100*RA/(RA+WA); "%).": RETURN
1370 IF R$ = FC$(N,S2) THEN 1500
1400 PRINT: PRINT "Sorry, it´s "; FC$(N,S2); "."
1410 WA = WA + 1: GOTO 1230
1500 PRINT: PRINT "That´s right!"
1510 RA = RA + 1: GOTO 1230
2000 PRINT CHR$(12); HD$: PRINT
2010 PRINT "New flashcards -- option #2": PRINT
2020 GOTO 3020
3000 PRINT CHR$(12); HD$: PRINT
3010 PRINT "Add flashcards -- option #3": PRINT
3020 PRINT: PRINT "Both sides of each flashcard ";
3030 PRINT "are stored in DATA statements."
3040 PRINT:PRINT"The two sides are to be put in ";
3050 PRINT "either one DATA statement,"
3060 PRINT "or in two successive DATA statements.";
```

```
3070 PRINT "  All items in the"
3080 PRINT "DATA statements must be enclosed in ";
3090 PRINT "quotation marks.": PRINT
3100 PRINT "Be sure that all entries USE ";
3110 PRINT "UPPERCASE letters!": PRINT
3120 PRINT "Number the data statements between ";
3130 W$ = "the last one after 6000      "
3140 IF R = 2 THEN W$ = "6000"
3150 PRINT W$; " and 9999, in order."
3160 PRINT: PRINT "After you've entered the new ";
3170 PRINT "DATA, input 'RUN' to either"
3180 PRINT "use the new set, or store it on tape."
3190 IF R = 3 THEN 3220
3200 PRINT: PRINT "To double-check that no DATA ";
3210 GOTO 3230
3220 PRINT: PRINT "So that you can see what DATA ";
3230 PRINT "statements are now used,"
3240 PRINT "a list will be printed.  When you're ";
3250 INPUT "ready, hit 'RETURN'"; R$: PRINT: PRINT
3260 LIST 5999: END
4000 PRINT CHR$(12); HD$: PRINT
4010 PRINT "Cassette save -- option #4": PRINT
4020 PRINT: PRINT CM$: PRINT: PRINT
4030 PRINT "Don't overwrite your original 'blank'";
4040 PRINT " version of the"
4050 PRINT "program, or any other version you ";
4060 PRINT "want to save.": PRINT
4070 PRINT "Store the ENTIRE program, complete ";
4080 PRINT "with your data statements,"
4090 PRINT "like any other program, using ";
4100 PRINT "'CSAVE XXXXX'."
4110 PRINT "The 'XXXXX' can be any 5-letter name."
4120 GOTO 3220
5000 PRINT: PRINT CM$: PRINT: PRINT
5010 PRINT "Have you stored the current set yet?"
5020 PRINT "If you forgot, enter '*MENU' and do ";
5030 PRINT "it now.  But if"
5040 PRINT "you REALLY want to quit now, just ";
5050 PRINT "hit 'RETURN'";: INPUT R$
5060 IF R$ = "*MENU" THEN RETURN
5999 END: REM -- put DATA in lines 6000-9999
10000 DATA "XXX","XXX": REM -- Keep stop flag here
10001 REM
```

```
10002 REM REMEMBER -- 1) Proper DATA format
10003 REM             2) Proper DATA line numbers
10004 REM             3) Use 'CSAVE' to save
10005 REM             4) Use 'RUN' to start over
```

EASY CHANGES

1. Change the limits of the number of flashcards that can be used by changing line 150. MX is the upper limit and MN is the minimum. The value following CLEAR may also need to be increased as well, if an ?/**OS** error occurs. Do not make MN much larger than 10, or the program will be appreciably slowed.
2. If you want to use some keyword other than ***MENU**, substitute whatever you like in lines 1330 and 5060.
3. To cause the program to always display side one of the flashcards (and ask the user for side two), insert this line:

   1275 S1 = 1: S2 = 2

   To cause it to always display side two, insert this:

   1275 S1 = 2: S2 = 1

MAIN ROUTINES

150-190 Initializes variables. Dimensions arrays. Displays heading.

200-280 Displays options menu, branches to correct subroutine.

1000-1220 Reads flashcards from DATA statements. Prints heading.

1230-1270 Selects front or back of card. Checks for recent repeats.

1300-1510 Prints one side of card and gets user's answer. Updates wrong or right totals, then repeats from line 1230.

2000-2020 Subroutine to display instructions for new flashcard entry.

3000-3260 Subroutine to display instructions for flashcard entry and list of current DATA statement contents.

4000-4120 Subroutine to display headings for saving cards on tape.

5000-5060	Subroutine to end program, after a reminder.
6000-9999	DATA statements with flashcard strings (in pairs).

MAIN VARIABLES

MX	Upper limit of flashcards that can be entered.
MN	Lower limit of flashcards that can be entered.
N	Subscript of random card chosen during drill.
NCARD	Number of cards read.
WA	Total of wrong responses.
RA	Total of right answers.
FC$	Array containing flashcard phrases.
PF	Array of subscripts of MN − 1 previous cards used in drill.
J	Loop and subscript variable.
CM$,HD$	Output strings.
S1,S2	Variables to select front or back of card.
R$,R	User's response. Also, option number selected.
W$,X$	Work string variables.

SUGGESTED PROJECTS

1. Modify the program for use in a classroom. You might want to allow only the drill option to be selected, with a fixed number of questions to be asked.
2. Modify the program to store flashcard phrases as ASCII values in a numeric array. This will permit phrases to be stored as separate data on tape, much as the mileage data is read and stored in the MILES program.

METRIC

PURPOSE

In case you don't realize it, we live in a metric world. The United States is one of the last holdouts, but that is changing rapidly. If you're still inching along or watching those pounds, it's time to convert.

METRC is an instructional program designed to familiarize you with the metric system. It operates in a quiz format with the program randomly creating questions from its data resources. You are then asked to compare two quantities—one in our old English units and one in the corresponding metric units. When you are wrong, the exact conversion and the rule governing it are given.

The two quantities to compare are usually within 50% of each other. Thus, you are constantly comparing an "English" quantity and a metric one which are in the same ball park. This has the effect of providing you with some insight by sheer familiarity with the questions.

HOW TO USE IT

The first thing the program does is ask you how many questions you would like to do for the session. Any value of one or higher is acceptable. A random value is input next.

The sample run shows how each question is formulated. A quantity in English units is compared with one in metric units. Either one may appear first in the question. Each quantity will have an integral value. The relating word ("longer," "hotter," "heavier," etc.) indicates what type of quantities are being compared.

There are three possible replies to each question. Entering **Y** or **N** means that you think the answer is either yes or no, respectively. Entering any other key indicates that you have no idea as to the correct answer.

If you answer the question correctly, you will be duly congratulated and the rule will be displayed. A wrong answer or a response of "no idea," however, will generate some diagnostic information. The first value used in the question will be shown converted to its exact equivalent in the corresponding units. Also, the rule governing the situation will be displayed. At the end of any question, the program will request that you hit **RETURN** to proceed to the next question.

The program will continue generating the requested number of questions. Before ending, it will show you how many correct answers you gave and your percentage correct.

SAMPLE RUN

```
METRIC

METRIC - ENGLISH CONVERSIONS

How many questions shall we do? 3

Give me a random number, please? 1560


Question # 1 of 3

Is 83 kilometers per hour faster than
 77 miles per hour? N

You say no -- and you're right!

************* The rule is ************

One kilometer per hour equals
    .62136 miles per hour.

****************************************

Hit 'RETURN' to continue? _ (RETURN hit)


Question # 2 of 3
```

```
Is 28 litres more than 8 gallons? X

No idea?

****************************************

28 litres equals
   7.39698 gallons.

************* The rule is *************

One litre equals
   .26417 gallons.

****************************************

Hit 'RETURN' to continue? _ (RETURN hit)

Question # 3 of 3

Is 37 feet longer than 9 meters? N

You say no -- but you're wrong!

****************************************

37 feet equals
   11.2776 meters.

************* The rule is *************

One foot equals
   .3048 meters.

****************************************

Hit 'RETURN' to continue? _ (RETURN hit)

You had 1 right answers, out of 3 questions.
That's a score of 33 percent.
```

PROGRAM LISTING

```
100 REM <METRC>
110 REM Metric educational
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Russ
150 DIM ES$(30), MS$(30), EP$(30), MP$(30)
160 DIM RD$(30), CF(30): RESTORE
170 READ Q$: IF Q$ = "XXX" THEN 200
```

```
180 NC = NC + 1: ES$(NC) = Q$: READ MS$(NC)
190 READ RD$(NC), CF(NC), EP$(NC), MP$(NC):GOTO 170
200 PRINT CHR$(12); "METRIC - ENGLISH CONVERSIONS"
210 PRINT: PRINT: PRINT "How many questions shall";
220 INPUT " we do";NQ: NQ=INT(NQ): IF NQ<1 THEN 210
230 PRINT:INPUT "Give me a random number, please";Q
240 Q=RND(-ABS(Q)): NR=0: FOR J=1 TO NQ: GOSUB 300
250 NEXT J: PRINT: PRINT: PRINT "You had"; NR;
260 PRINT "right answers, out of"; NQ; "questions."
270 PRINT "That's a score of"; INT(100*NR/NQ);
280 PRINT "percent.": PRINT: PRINT: PRINT: END
300 N = INT(NC * RND(1) + 0.5): IF N < 1 THEN 300
310 PRINT CHR$(12): PRINT "Question #"; J; "of"; NQ
320 PRINT: PRINT: F = 0: IF RND(1) > 0.5 THEN F = 1
330 VL = INT(98*RND(1) + 1): VC = VL * CF(N)
340 IF F = 1 THEN VC = VL / CF(N)
350 IF N = 1 THEN VC = (VL - 32) / 1.8
360 IF N = 1 AND F = 1 THEN VC = (VL * 1.8) + 32
370 VR = INT(1.5 + VC * (0.5 + RND(1)))
380 T = 0: IF VR < VC THEN T = 1
400 IF F = 1 THEN 430
410 PRINT "Is"; VL; EP$(N); " "; RD$(N); " than";
420 PRINT VR; MP$(N);: GOTO 450
430 PRINT "Is"; VL; MP$(N); " "; RD$(N); " than";
440 PRINT VR; EP$(N);
450 Q$ = "": INPUT Q$: R = 2: PRINT: PRINT
455 IF Q$ = "Y" OR Q$ = "y" THEN R = 1
460 IF Q$ = "N" OR Q$ = "n" THEN R = 0
465 ON R GOTO 480, 490
470 PRINT "You say no -- ";: GOTO 500
480 PRINT "You say yes -- ";: GOTO 500
490 PRINT "No idea?": GOSUB 800: RETURN
500 IF T - R = 0 THEN 520
510 PRINT "but you're wrong!": GOSUB 800: RETURN
520 PRINT "and you're right!": NR = NR + 1
530 GOSUB 870: RETURN
800 PRINT: PRINT
810 PRINT "*******************************************"
820 PRINT: IF F = 1 THEN 850
830 PRINT VL; EP$(N); " equals"
840 PRINT "     "; VC; MP$(N); ".": GOTO 870
850 PRINT VL; MP$(N); " equals"
860 PRINT "     "; VC; EP$(N); "."
870 PRINT
```

```
880 PRINT "************* The rule is *************"
890 PRINT: IF N > 1 THEN 960
900 IF F = 1 THEN 930
910 PRINT "Degrees centigrate ="
920 PRINT"    (degrees fahrenheit - 32)/1.8":GOTO990
930 PRINT "Degrees fahrenheit ="
940 PRINT "   (degrees centigrade * 1.8) + 32"
950 GOTO 990
960 IF F = 1 THEN 980
970 PRINT "One "; ES$(N); " equals"
975 PRINT "     "; CF(N); MP$(N); ".": GOTO 990
980 PRINT "One "; MS$(N); " equals"
985 Q=INT(1E5/CF(N))/1E5: PRINT "     ";Q;EP$(N);"."
990 PRINT
992 PRINT "*****************************************"
995 PRINT: INPUT "Hit 'RETURN' to continue"; Q$
999 RETURN
1000 DATA "degree fahrenheit", "degree centigrade"
1010 DATA "hotter", 0.5, "degrees fahrenheit"
1020 DATA "degrees centigrade"
1030 DATA "mile per hour", "kilometer per hour"
1040 DATA "faster", 1.60935, "miles per hour"
1050 DATA "kilometers per hour"
1060 DATA "foot", "meter", "longer", 0.3048
1070 DATA "feet", "meters"
1090 DATA "mile", "kilometer", "longer", 1.60935
1100 DATA "miles", "kilometers"
1110 DATA "inch", "centimeter", "longer", 2.54
1120 DATA "inches", "centimeters"
1130 DATA "gallon", "litre", "more", 3.78533
1140 DATA "gallons", "litres"
1150 DATA "pound", "kilogram", "more massive"
1160 DATA 0.45359, "pounds", "kilograms"
9999 DATA "XXX"
```

EASY CHANGES

1. To have the program always ask a fixed number of questions, remove line 220 and change line 210 to set NQ to the desired value. For example:

   210 NQ=10

   will cause the program to do 10 questions.

2. There are currently seven conversions built into the program:

N	*Type*	*English Unit*	*Metric Unit*
1	temperature	degrees F	degrees C
2	speed	miles/hour	kilometers/hour
3	length	feet	meters
4	length	miles	kilometers
5	length	inches	centimeters
6	volume	gallons	liters
7	mass (weight)	pounds	kilograms

   If you wish to be quizzed on only one type of question, set N to this value in line 300. Thus,

```
300 N=4
```

   will cause the program to only produce questions concerning miles and kilometers. To add additional data to the program, see the first "Suggested Project."

3. You can easily have the questions posed in one "direction" only. To go only from English to metric units use

```
320 PRINT:PRINT:F=0
```

   while to go from metric to English units use

```
320 PRINT:PRINT:F=1
```

4. You might want both the converted value and governing rule to be displayed even when the correct answer is given. This is accomplished by changing line 530 as follows:

```
530 GOSUB 800:RETURN
```

MAIN ROUTINES

150-190	Dimensions and initializes arrays.
200-280	Mainline routine, drives other routines.
300-530	Forms and asks questions. Processes user's reply.
800-999	Displays exact conversion and governing rule.
1000-9999	Data statements.

MAIN VARIABLES

NC	Number of conversions in the data.
ES$,EP$	String arrays of English units' names (singular, plural).

MS$,MP$	String arrays of metric units' names (singular, plural).
RD$	String array of the relation descriptors.
CF	Array of the conversion factors.
Q	Work variable.
J	Current question number.
NR	Number of questions answered right.
NQ	Number of questions in session.
N	Index number of current question in the data list.
F	Flag on question "direction" (0=English to metric; 1=metric to English).
VL,VR	Numeric values on left, right sides of the question.
VC	The correct value of the right-hand side.
T	Flag on the question's correct answer (1=true; 0=false).
Q$	User reply string.
R	User reply flag (0=no; 1=yes; 2=no idea).

SUGGESTED PROJECTS

1. Each built-in conversion requires six elements of data in this order:

Element	*Data Description*
1	English unit (singular)
2	Metric unit (singular)
3	Relation descriptor (e.g. "hotter," "faster," etc.)
4	Conversion factor (from English to metric)
5	English unit (plural)
6	Metric unit (plural)

Each of these elements, except the fourth, is a string. The data statements in the listing should make clear how the information is to be provided. You can add new data to the program with appropriate data statements in this format. New data should be added after the current data, i.e. just before line 9999. Line 9999 is a special data statement to trigger the end of all data to the program. The program is dimensioned for up to 30 entries while only seven are currently used. (Note: this format allows only conversions where one unit is a direct multiple of the other. Temperature, which does not fit this rule, is handled as a special case throughout the program.)

2. Convert the program to handle units conversion questions of any type.
3. Keep track of the questions asked and which ones were missed. Then do not ask the same questions too soon if they have been answered correctly. However, repeat the missed questions to provide additional practice.

NUMBERS

PURPOSE

This is an educational program for preschool children. After a few weeks of watching Sesame Street on television, most 3- and 4-year-old children will learn how to count from one to ten. NUMBR allows these children to practice their numbers and to have fun at the same time.

HOW TO USE IT

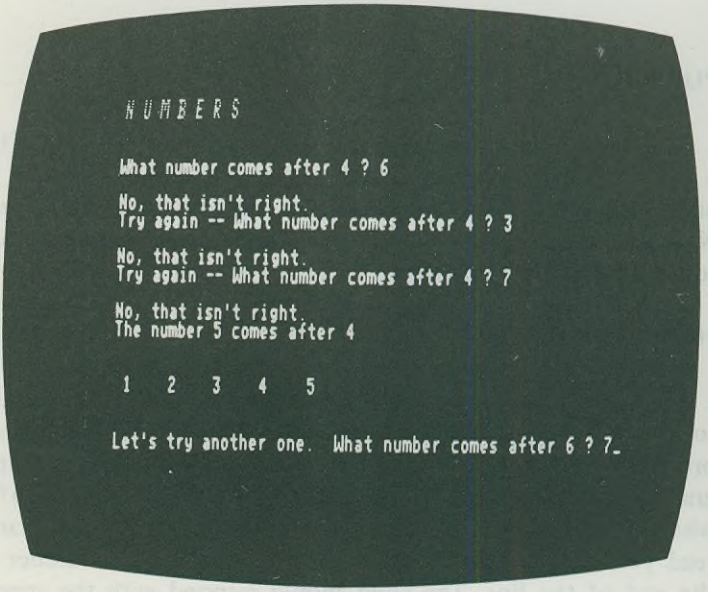
We know a child who learned how to type **CLOAD** and **RUN** to get this program started before she turned three, but you'll probably have to help your child with this for a while. The program asks the question, "WHAT NUMBER COMES AFTER n?", where n is a number from one to nine. Even if the child can't read yet, he or she will soon learn to look for the number at the end of the line. The child should respond with the appropriate number, and then press the **RETURN** key.

If the answer is correct, the program displays the message "That's right!", pauses for a couple of seconds, then clears the screen and displays three geometric shapes. In the upper left of the screen a square is drawn. In the lower center, a triangle is drawn. Then, an asterisk (or a snowflake, perhaps?) is drawn in the right portion of the screen. After a delay, the program clears the screen and asks another question. The same number is never asked twice in a row. The size of the three figures is chosen at random each time. Also, the characters used in drawing the shapes are chosen at random (from a few possibilities) each time.

If the child provides the wrong answer, a message indicates the error and the same question is asked again, up to three times.

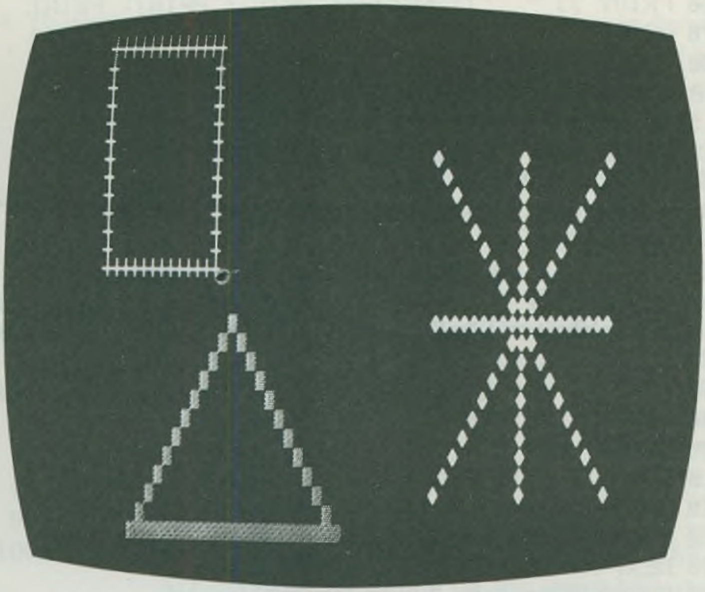
The program keeps on going until you hit **CTRL C**. Remember that most children have a pretty short attention span, so please do not force your child to continue after his or her interest diminishes. Keep each session short and fun. This way, it will always be a treat to "play" with the computer.

SAMPLE RUN



```
N U M B E R S

What number comes after 4 ? 6

No, that isn't right.
Try again -- What number comes after 4 ? 3

No, that isn't right.
Try again -- What number comes after 4 ? 7

No, that isn't right.
The number 5 comes after 4


 1   2   3   4   5


Let's try another one.  What number comes after 6 ? 7_
```

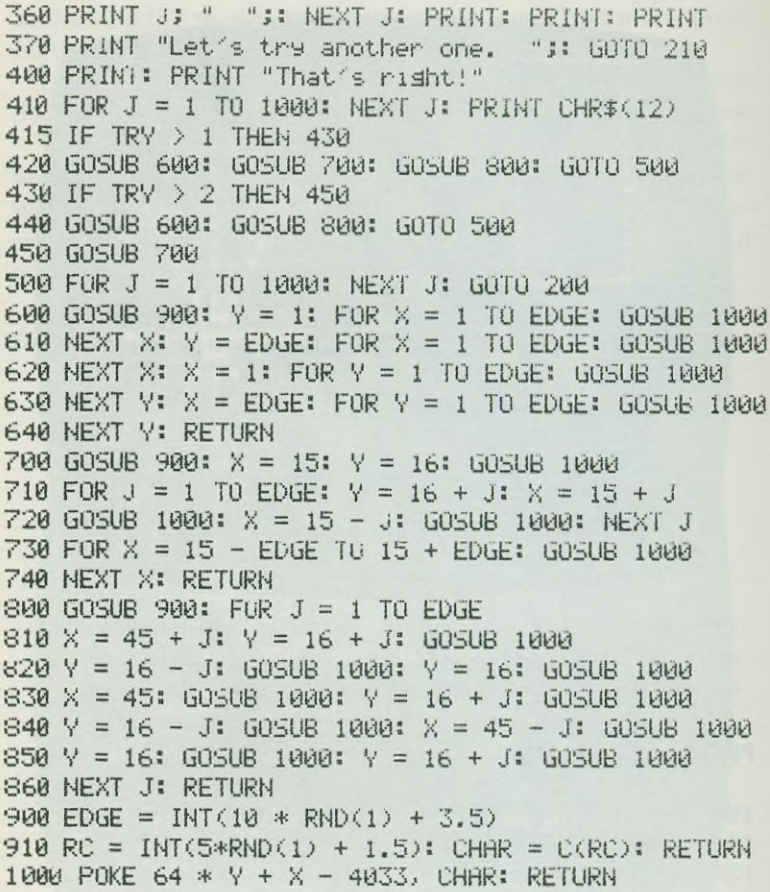
The first question asks for the number that comes after 4. After three wrong replies, a "number line" is shown and a new question is asked.



After a correct response, geometric figures are drawn, using random characters. If the first answer given was correct, three figures are drawn; if several tries were required, fewer figures will appear.

PROGRAM LISTING

```
100 REM <NUMBR>
110 REM Counting drill
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 DATA 132, 136, 142, 163, 173, 177
160 NMAX = 9: R = RND(-1)
170 DIM C(6): FOR J = 1 TO 6: READ C(J): NEXT J
200 PRINT CHR$(12); "N U M B E R S": PRINT: PRINT
210 PREV = R: TRY = 0
220 R = INT((NMAX-1)*RND(1)) + 1:IF R=PREV THEN 220
300 TRY = TRY + 1: PRINT "What number comes after";
310 PRINT R;: INPUT AN$: IF VAL(AN$)=R+1 THEN 400
315 IF VAL(AN$) = R + 1 THEN 400
320 PRINT: PRINT "No, that isn't right."
330 IF TRY < 3 THEN PRINT "Try again -- ";:GOTO 300
340 PRINT "The number"; R+1; "comes after"; R:PRINT
350 PRINT: FOR J = 1 TO R + 1
```



```
360 PRINT J; "  ";: NEXT J: PRINT: PRINT: PRINT
370 PRINT "Let's try another one.  ";: GOTO 210
400 PRINT: PRINT "That's right!"
410 FOR J = 1 TO 1000: NEXT J: PRINT CHR$(12)
415 IF TRY > 1 THEN 430
420 GOSUB 600: GOSUB 700: GOSUB 800: GOTO 500
430 IF TRY > 2 THEN 450
440 GOSUB 600: GOSUB 800: GOTO 500
450 GOSUB 700
500 FOR J = 1 TO 1000: NEXT J: GOTO 200
600 GOSUB 900: Y = 1: FOR X = 1 TO EDGE: GOSUB 1000
610 NEXT X: Y = EDGE: FOR X = 1 TO EDGE: GOSUB 1000
620 NEXT X: X = 1: FOR Y = 1 TO EDGE: GOSUB 1000
630 NEXT Y: X = EDGE: FOR Y = 1 TO EDGE: GOSUB 1000
640 NEXT Y: RETURN
700 GOSUB 900: X = 15: Y = 16: GOSUB 1000
710 FOR J = 1 TO EDGE: Y = 16 + J: X = 15 + J
720 GOSUB 1000: X = 15 - J: GOSUB 1000: NEXT J
730 FOR X = 15 - EDGE TO 15 + EDGE: GOSUB 1000
740 NEXT X: RETURN
800 GOSUB 900: FOR J = 1 TO EDGE
810 X = 45 + J: Y = 16 + J: GOSUB 1000
820 Y = 16 - J: GOSUB 1000: Y = 16: GOSUB 1000
830 X = 45: GOSUB 1000: Y = 16 + J: GOSUB 1000
840 Y = 16 - J: GOSUB 1000: X = 45 - J: GOSUB 1000
850 Y = 16: GOSUB 1000: Y = 16 + J: GOSUB 1000
860 NEXT J: RETURN
900 EDGE = INT(10 * RND(1) + 3.5)
910 RC = INT(5*RND(1) + 1.5): CHAR = C(RC): RETURN
1000 POKE 64 * Y + X - 4033, CHAR: RETURN
```

EASY CHANGES

1. Change the range of numbers that the program asks by altering the value of NMAX in line 160. For a beginner, use a value of 3 for NMAX instead of 9. Later, increase the value to 5, and then 15.
2. Alter the delay after “That's right!” is displayed by altering the value of 1000 in statement 410. Double it to double the time delay, etc. The same can be done with the 1000 in line 500 to alter the delay after the figures are drawn.
3. To avoid randomness in the size of the figures that are drawn, replace line 900 with

   900 EDGE=10

Instead of 10, you can use any integer from 3 to 13.

4. To use a wider range of random graphics characters in drawing the shapes, increase the dimension and loop constant of 6 in line 170, and the constant 5 in line 910. Add additional ASCII values to the DATA list in line 150.

MAIN ROUTINES

150-170	Initializes variables.
200-220	Picks random integer from 1 to NMAX.
300-370	Asks question. Gets answer. Determines if right or wrong.
400-500	Selects and draws shapes.
600-640	Draws a square.
700-740	Draws a triangle.
800-860	Draws an asterisk.
900-910	Selects size and character for a shape.
1000	Subroutine to POKE graphics character CHAR to X,Y coordinate location on the screen (measured from top left).

MAIN VARIABLES

NMAX	Maximum number that will be asked.
EDGE	Edge length of geometric figures.
R	Random integer in range from 1 to NMAX.
PREV	Previous number that was asked.
AN$	Reply given by operator.
X,Y	Coordinates in CRT display.
CHAR	Graphics character to be POKEd to CRT screen.
RC	Subscript variable of output character.
C	Array of ASCII graphic character values.
TRY	Number of tries.

SUGGESTED PROJECTS

1. Modify the program to ask the next letter of the alphabet. Use the ASC and CHR$ functions in picking a random letter from A to Y, and to check whether the response is correct or not.

2. Ask each number from 1 to NMAX once (in a random sequence). At the end of the sequence, repeat those that were missed.
3. Add different shapes to the graphics display that is shown after a correct answer. Try an octagon, a diamond, and a rectangle, or combine this program with one of the graphics display programs.

TACHIST

PURPOSE

This program turns your computer into a tachistoscope (tah-KISS-tah-scope). A tachistoscope is used in reading classes to improve reading habits and, as a result, improve reading speed. The program displays a word or phrase on the screen for a fraction of a second, then asks you what it was. With a little practice, you will find that you can read phrases that are displayed for shorter and shorter time periods.

HOW TO USE IT

The program begins by displaying a brief introduction, then waiting for a random value to be input. The screen will then be blanked, except for two horizontal dashed lines in the center. After a pause, a phrase is flashed on the screen between the two lines, then the whole screen is blanked.

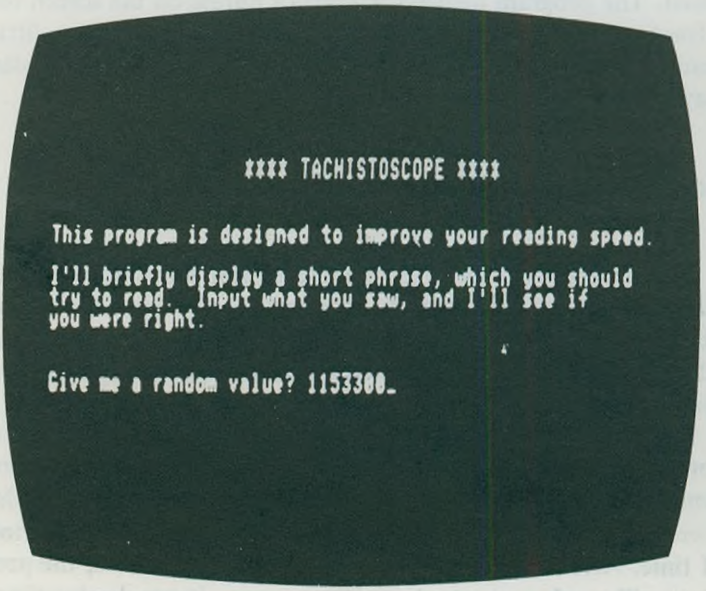
Next the user is asked what the phrase was. If the response is correct, the next phrase is displayed for about half the prior time period. If the response is wrong, the program shows the correct phrase, then displays the next phrase for a longer period of time. After a long series of right or wrong answers, the program will reach a constant minimum or maximum display time, respectively.

A great deal of research has been done to determine how to help people to read faster and with better comprehension. See the bibliography for further information—it's a complex problem. However, there is one method that seems to work for most readers, and the Sorcerer's program is designed to incorporate it.

A fast reader reads more than one word at a time. Most "speed reading" courses use this as their basic principle. Instead of looking at each word individually, focus your eyes at a point near the center of a phrase. No, it's *not* more difficult to read three or five or more words at a glance. Your brain is fast enough, but the slow word-by-word scan of your eyes has always added delays.

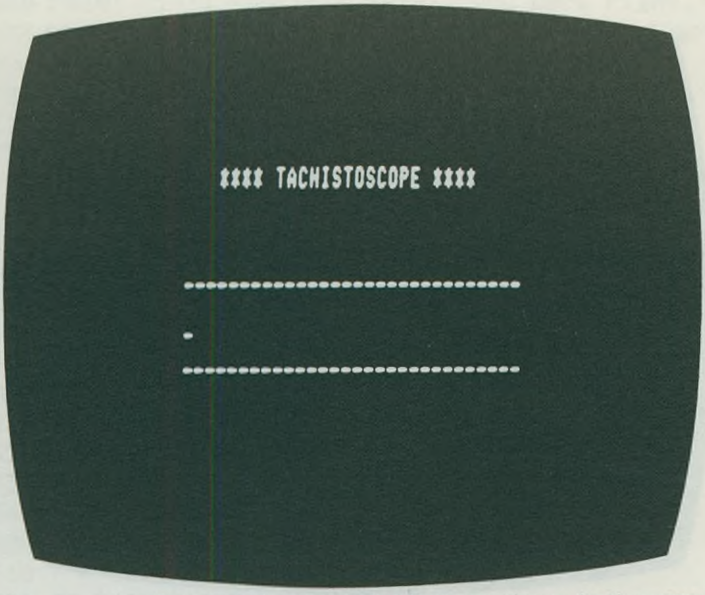
Because the tachistoscope flashes an entire phrase on the screen almost instantaneously, it forces you to look at a single point. There just isn't time for a slow scan. After some practice, you should be able to read each phrase at the machine's fastest speed.

SAMPLE RUN



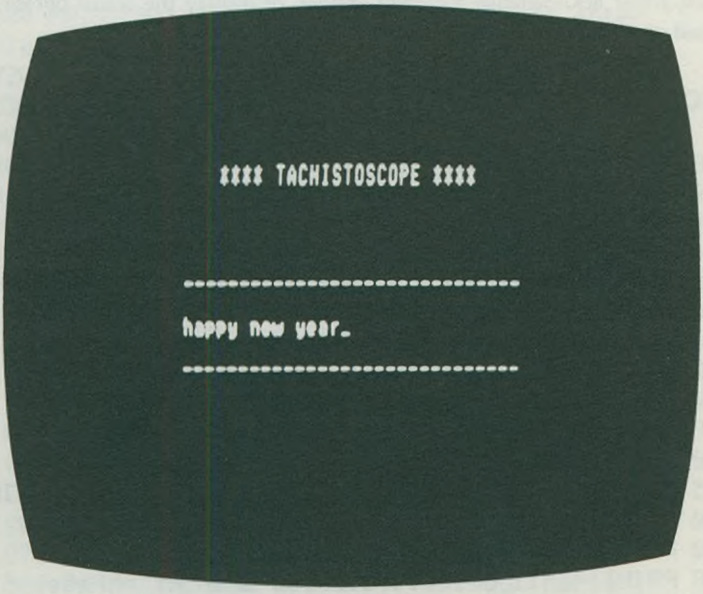
```
                **** TACHISTOSCOPE ****

This program is designed to improve your reading speed.

I'll briefly display a short phrase, which you should
try to read.  Input what you saw, and I'll see if
you were right.


Give me a random value? 1153300_
```

The program displays an introduction, then waits for a random value to be input.



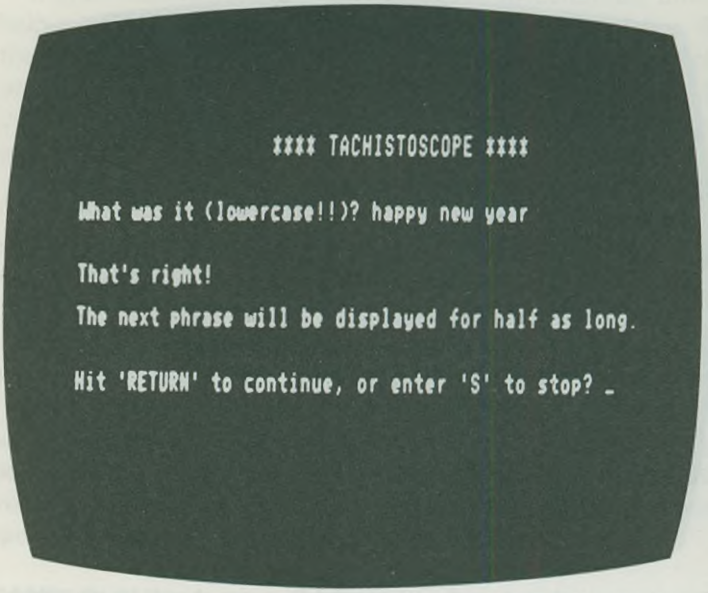
```
**** TACHISTOSCOPE ****


-------------------------------
-
-------------------------------
```

The screen is cleared, and two parallel lines are shown in the upper center.



```
**** TACHISTOSCOPE ****


-------------------------------
happy new year._
-------------------------------
```

A randomly selected short phrase is flashed on the screen between the parallel lines. After a brief pause, the screen is cleared again.

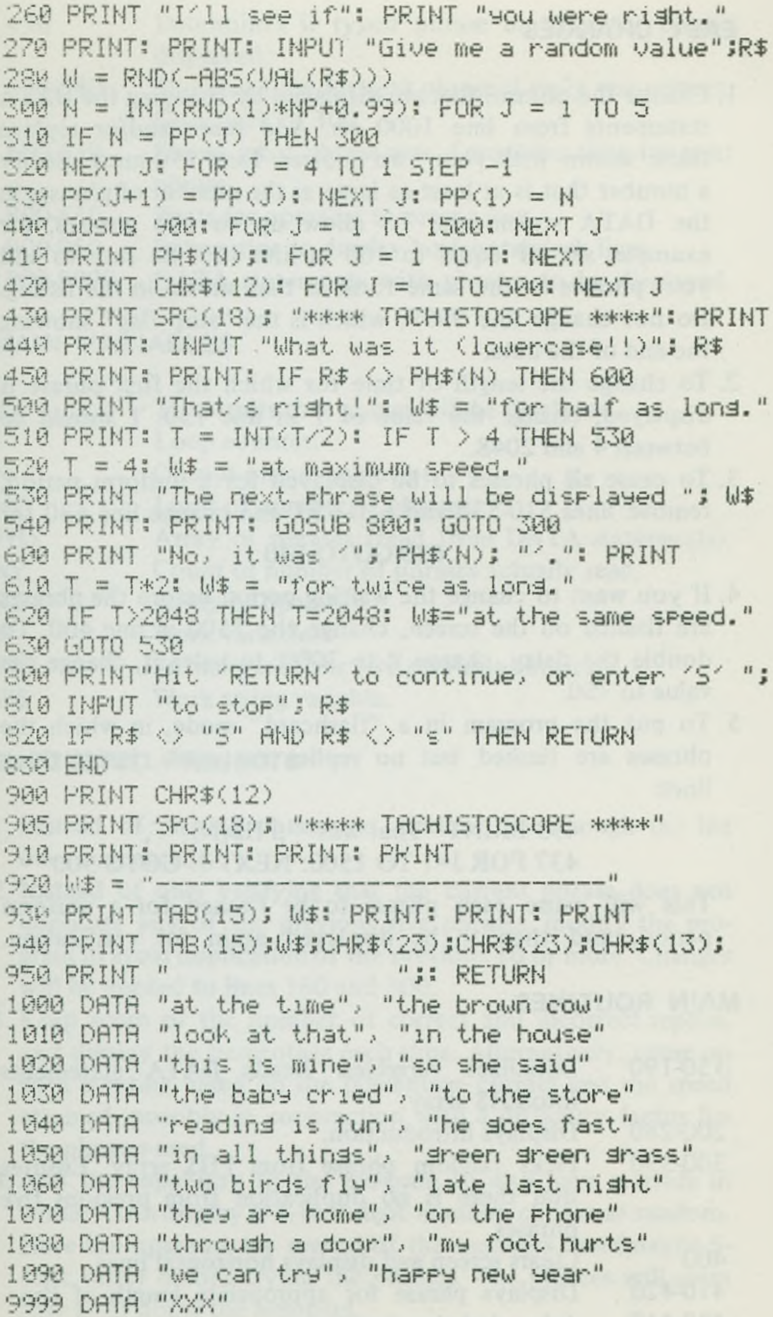


```
                    **** TACHISTOSCOPE ****

What was it (lowercase!!)? happy new year

That's right!
The next phrase will be displayed for half as long.

Hit 'RETURN' to continue, or enter 'S' to stop? _
```

The user inputs what he saw, using lowercase letters. The program will either acknowledge a correct answer, or display the actual phrase used, then adjust the display time for the next attempt.

PROGRAM LISTING

```
100 REM <TACHI>
110 REM Tachistoscopic reading
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 CLEAR 2000: MP = 50: NP = 0: T = 128
160 DIM PH$(MP), PP(5): RESTORE
170 READ W$: IF W$ = "XXX" THEN 200
180 NP=NP+1: IF NP<=50 THEN PH$(NP)=W$: GOTO 170
190 PRINT "RUN ABORTED -- too many phrases!": END
200 PRINT CHR$(12)
205 PRINT SPC(18); "**** TACHISTOSCOPE ****": PRINT
210 PRINT: PRINT "This program is designed to ";
220 PRINT "improve your reading speed.": PRINT
230 PRINT "I'll briefly display a short phrase, ";
240 PRINT "which you should"
250 PRINT "try to read.  Input what you saw, and ";
```



```
260 PRINT "I'll see if": PRINT "you were right."
270 PRINT: PRINT: INPUT "Give me a random value";R$
280 W = RND(-ABS(VAL(R$)))
300 N = INT(RND(1)*NP+0.99): FOR J = 1 TO 5
310 IF N = PP(J) THEN 300
320 NEXT J: FOR J = 4 TO 1 STEP -1
330 PP(J+1) = PP(J): NEXT J: PP(1) = N
400 GOSUB 900: FOR J = 1 TO 1500: NEXT J
410 PRINT PH$(N);: FOR J = 1 TO T: NEXT J
420 PRINT CHR$(12): FOR J = 1 TO 500: NEXT J
430 PRINT SPC(18); "**** TACHISTOSCOPE ****": PRINT
440 PRINT: INPUT "What was it (lowercase!)"; R$
450 PRINT: PRINT: IF R$ <> PH$(N) THEN 600
500 PRINT "That's right!": W$ = "for half as long."
510 PRINT: T = INT(T/2): IF T > 4 THEN 530
520 T = 4: W$ = "at maximum speed."
530 PRINT "The next phrase will be displayed "; W$
540 PRINT: PRINT: GOSUB 800: GOTO 300
600 PRINT "No, it was '"; PH$(N); "'.": PRINT
610 T = T*2: W$ = "for twice as long."
620 IF T>2048 THEN T=2048: W$="at the same speed."
630 GOTO 530
800 PRINT"Hit 'RETURN' to continue, or enter 'S' ";
810 INPUT "to stop"; R$
820 IF R$ <> "S" AND R$ <> "s" THEN RETURN
830 END
900 PRINT CHR$(12)
905 PRINT SPC(18); "**** TACHISTOSCOPE ****"
910 PRINT: PRINT: PRINT: PRINT
920 W$ = "-----------------------------"
930 PRINT TAB(15); W$: PRINT: PRINT: PRINT
940 PRINT TAB(15);W$;CHR$(23);CHR$(23);CHR$(13);
950 PRINT "               ";: RETURN
1000 DATA "at the time", "the brown cow"
1010 DATA "look at that", "in the house"
1020 DATA "this is mine", "so she said"
1030 DATA "the baby cried", "to the store"
1040 DATA "reading is fun", "he goes fast"
1050 DATA "in all things", "green green grass"
1060 DATA "two birds fly", "late last night"
1070 DATA "they are home", "on the phone"
1080 DATA "through a door", "my foot hurts"
1090 DATA "we can try", "happy new year"
9999 DATA "XXX"
```

EASY CHANGES

1. Change the phrases that are displayed by changing the DATA statements from line 1000 on. Add more and/or replace those shown with your own choices. Line 150 must specify a number that is at least as large as the number of phrases in the DATA statements. To allow up to 100 phrases, for example, set MP equal to 100 in line 150. Be sure to add your phrases in the same form as that shown in the listing. Do not change line 9999, which is the "stop flag" showing the end of the data.
2. To change the length of time for which the first phrase is displayed, change the value of T in line 150. T should be between 4 and 2048.
3. To cause all phrases to be displayed for a uniform period, remove lines 510-530 and 610-620, and change line 630 to:

   ```
   630 GOTO 540
   ```
4. If you want to change the waiting period before the phrases are flashed on the screen, change the 1500 in line 400. To double the delay, change it to 3000; to halve it, change the value to 750.
5. To put the program in a "flashcard" mode, in which the phrases are flashed but no replies are made, insert these lines:

   ```
   433 PRINT “That was ’”;PH$(N); “‘,”
   437 FOR J=1 TO 1500: NEXT J: GOTO 300
   ```

   This will cause each phrase to be flashed for a uniform period, and then to be displayed again for verification.

MAIN ROUTINES

150-190	Initializes variables. Reads DATA statements into PH$ array.
200-280	Displays introduction.
300-330	Picks random phrase from PH$ array. Ensures that there is no duplication from previous five phrases.
400	Clears screen and displays horizontal lines.
410-420	Displays phrase for appropriate length of time.
430-440	Asks what the phrase was.

450	Determines if typed phrase matches the phrase displayed.
500-540	Shortens time for next phrase if reply was correct. Goes back to 300.
600-630	Shows what phrase was. Lengthens time for next phrase.
800-830	Waits for operator to press a key.
900-950	Subroutine to display horizontal dash lines.
1000-9999	DATA statements with phrases to be displayed.

MAIN VARIABLES

T	Time units that phrase will be displayed.
J	Loop counter.
N	Output phrase subscript.
MP	Limit of number of phrases.
PH$	Array of phrases (read from DATA statements).
NP	Count of number of phrases actually read.
R$	Reply of operator.
W	Work variable.
PP	Subscripts of the five previous phrases.
W$	Work string variable.

SUGGESTED PROJECTS

1. Instead of picking phrases at random, go through the list once sequentially.
2. Instead of only verifying that the current phrase does not duplicate any of the previous five phrases, modify the program to avoid duplication of the previous 10 or more. Changes will be needed to lines 160 and 300.
3. Keep score of the number of correct and incorrect replies, and display the percentage each time. Alternatively, come up with a rating based on the percentage correct and the speed attained, possibly in conjunction with a difficulty factor for the phrases used.
4. Add the capability to the program to also have a mode in which it can display a 2- to 7-digit number, chosen at random. Have the operator try several of the numbers first (maybe 5-digit ones) before trying the phrases. The phrases will seem easy after doing the numbers.

VOCAB

PURPOSE

Did you ever find yourself at a loss for words? This vocabulary quiz can be used in a self-teaching environment or as reinforcement for classroom instruction to improve your ability to remember the jargon of any subject. It allows you to drill at your own pace, without the worry of ridicule from other students or judgment by an instructor. When you make mistakes, only the computer knows, and it's not telling anyone except you. Modifying the program to substitute a different vocabulary list is very simple, so you can accumulate many different versions of this program, each with a different set of words.

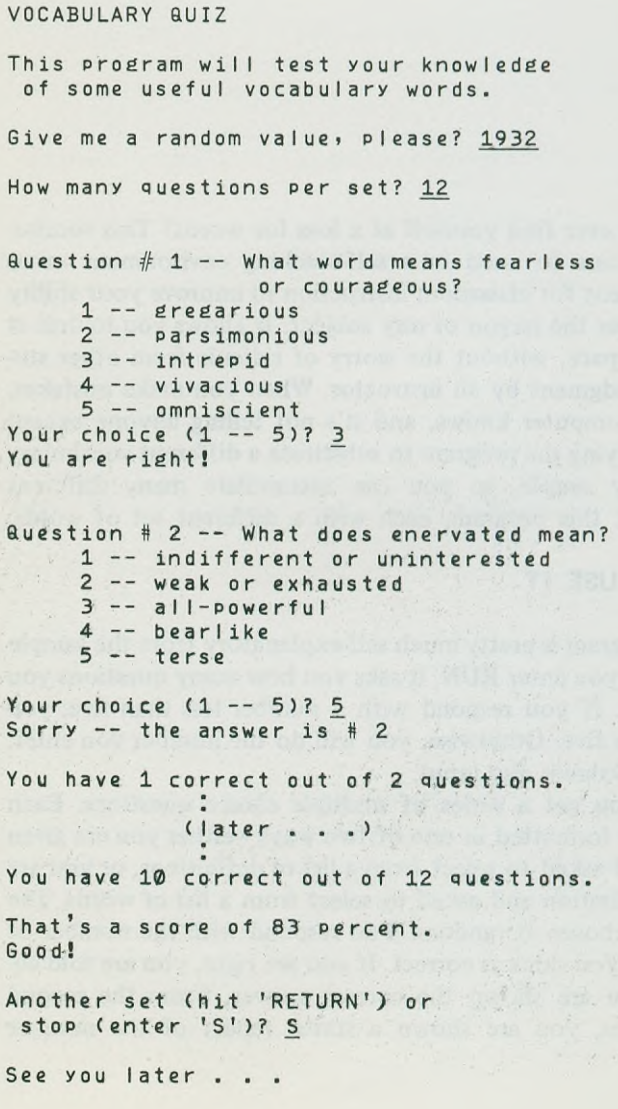
HOW TO USE IT

This program is pretty much self-explanatory from the sample run. After you enter **RUN**, it asks you how many questions you would like. If you respond with a number less than five, you will still do five. Otherwise, you will do the number you enter. A random value is also input.

Next, you get a series of multiple choice questions. Each question is formatted in one of two ways—either you are given a word and asked to select from a list of definitions, or you are given a definition and asked to select from a list of words. The format is chosen at random. You respond with the number of the choice you think is correct. If you are right, you are told so. If not, you are shown the correct answer. From the second question on, you are shown a status report of the number correct out of the number attempted so far.

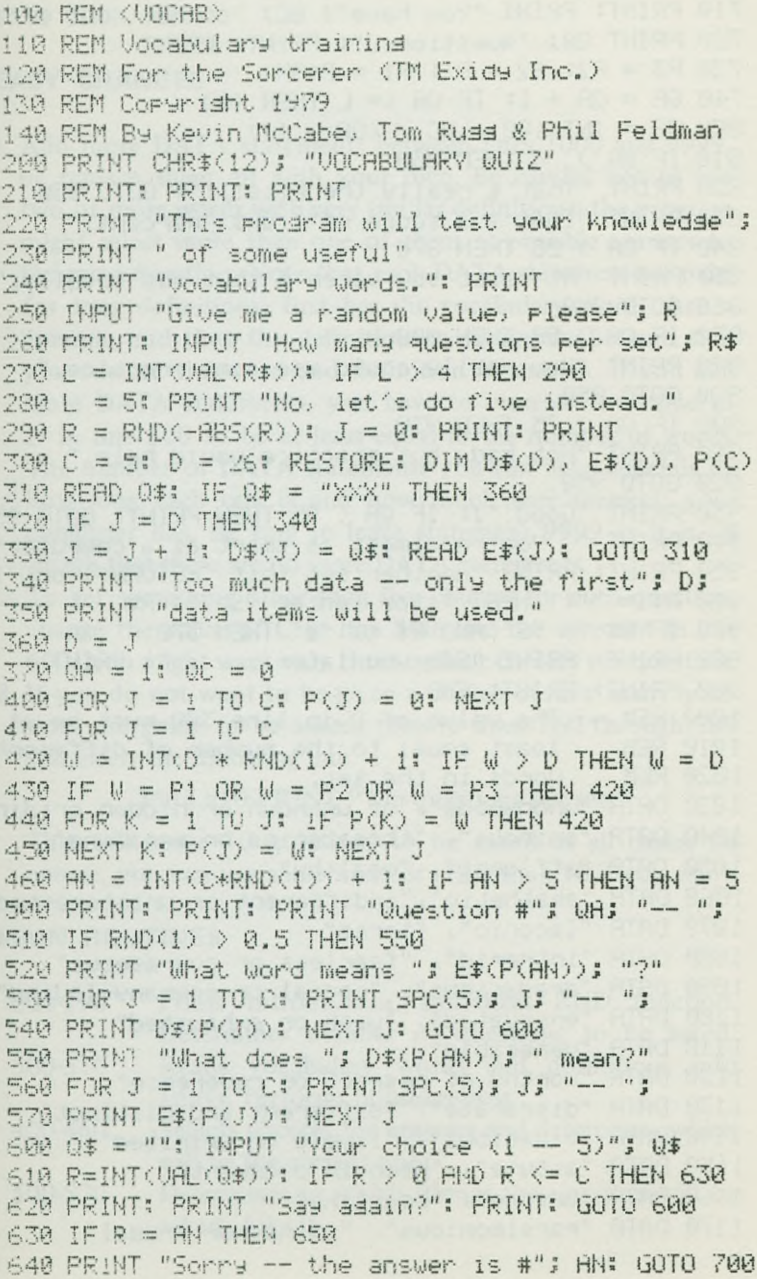
Finally, after the last question, you are shown the percentage correct, along with a comment on your performance. Then you have the option of going back for another round of questions, or stopping.

SAMPLE RUN



```
VOCABULARY QUIZ

This program will test your knowledge
 of some useful vocabulary words.

Give me a random value, please? 1932

How many questions per set? 12


Question # 1 -- What word means fearless
                  or courageous?
     1 -- gregarious
     2 -- parsimonious
     3 -- intrepid
     4 -- vivacious
     5 -- omniscient
Your choice (1 -- 5)? 3
You are right!

Question # 2 -- What does enervated mean?
     1 -- indifferent or uninterested
     2 -- weak or exhausted
     3 -- all-powerful
     4 -- bearlike
     5 -- terse

Your choice (1 -- 5)? 5
Sorry -- the answer is # 2

You have 1 correct out of 2 questions.
                       :
             (later . . .)
                       :
You have 10 correct out of 12 questions.

That's a score of 83 percent.
Good!

Another set (hit 'RETURN') or
 stop (enter 'S')? S

See you later . . .
```

PROGRAM LISTING



```
100 REM <VOCAB>
110 REM Vocabulary training
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
200 PRINT CHR$(12); "VOCABULARY QUIZ"
210 PRINT: PRINT: PRINT
220 PRINT "This program will test your knowledge";
230 PRINT " of some useful"
240 PRINT "vocabulary words.": PRINT
250 INPUT "Give me a random value, please"; R
260 PRINT: INPUT "How many questions per set"; R$
270 L = INT(VAL(R$)): IF L > 4 THEN 290
280 L = 5: PRINT "No, let's do five instead."
290 R = RND(-ABS(R)): J = 0: PRINT: PRINT
300 C = 5: D = 26: RESTORE: DIM D$(D), E$(D), P(C)
310 READ Q$: IF Q$ = "XXX" THEN 360
320 IF J = D THEN 340
330 J = J + 1: D$(J) = Q$: READ E$(J): GOTO 310
340 PRINT "Too much data -- only the first"; D;
350 PRINT "data items will be used."
360 D = J
370 QA = 1: QC = 0
400 FOR J = 1 TO C: P(J) = 0: NEXT J
410 FOR J = 1 TO C
420 W = INT(D * RND(1)) + 1: IF W > D THEN W = D
430 IF W = P1 OR W = P2 OR W = P3 THEN 420
440 FOR K = 1 TO J: IF P(K) = W THEN 420
450 NEXT K: P(J) = W: NEXT J
460 AN = INT(C*RND(1)) + 1: IF AN > 5 THEN AN = 5
500 PRINT: PRINT: PRINT "Question #"; QA; "-- ";
510 IF RND(1) > 0.5 THEN 550
520 PRINT "What word means "; E$(P(AN)); "?"
530 FOR J = 1 TO C: PRINT SPC(5); J; "-- ";
540 PRINT D$(P(J)): NEXT J: GOTO 600
550 PRINT "What does "; D$(P(AN)); " mean?"
560 FOR J = 1 TO C: PRINT SPC(5); J; "-- ";
570 PRINT E$(P(J)): NEXT J
600 Q$ = "": INPUT "Your choice (1 -- 5)"; Q$
610 R=INT(VAL(Q$)): IF R > 0 AND R <= C THEN 630
620 PRINT: PRINT "Say again?": PRINT: GOTO 600
630 IF R = AN THEN 650
640 PRINT "Sorry -- the answer is #"; AN: GOTO 700
```

```
650 PRINT "You are right!": QC = QC + 1
700 IF QA = 1 THEN 740
710 PRINT: PRINT "You have"; QC; "correct out of";
720 PRINT QA; "questions.": PRINT: PRINT
730 P3 = P2: P2 = P1: P1 = P(AN)
740 QA = QA + 1: IF QA <= L THEN 400
800 QA = INT(100 * QC / (QA - 1))
810 IF QA >  0 THEN 830
820 PRINT "That's really the pits!!!": GOTO 950
830 PRINT "That's a score of"; QA; "percent."
840 IF QA > 25 THEN 870
850 PRINT "At least you weren't shut out totally."
860 GOTO 950
870 IF QA > 50 THEN 900
880 PRINT "You really could use some practice."
890 GOTO 950
900 IF QA > 75 THEN 930
910 PRINT "Not bad, but practice would help."
920 GOTO 950
930 PRINT "Good!";: IF QA < 95 THEN PRINT: GOTO 950
940 PRINT "  You're almost as smart as a computer!"
950 PRINT: PRINT "Another set (hit 'RETURN') or ";
960 R$ = "": INPUT "stop (enter 'S')"; R$
970 IF R$ <> "S" AND R$ <> "s" THEN 370
980 PRINT: PRINT "See you later . . . ": PRINT
990 PRINT: PRINT: END
1000 REM -- The value of D in line 300 must be at
1010 REM    least equal to the number of different
1020 REM    words in the set.
1030 DATA "anonymous","of unknown or hidden origin"
1040 DATA "ominous", "threatening or menacing"
1050 DATA "affluent", "wealthy"
1060 DATA "apathetic","indifferent or uninterested"
1070 DATA "laconic", "terse"
1080 DATA "intrepid", "fearless or courageous"
1090 DATA "gregarious", "social or company-loving"
1100 DATA "enervated", "weak or exhausted"
1110 DATA "venerable"
1120 DATA "worthy of respect or reverence"
1130 DATA "disparate", "different and distinct"
1140 DATA "vivacious", "lively or spirited"
1150 DATA "astute", "keen in judgment"
1160 DATA "ursine", "bearlike"
1170 DATA "parsimonious", "stingy or frugal"
```

```
1180 DATA "omnicient", "all-knowing"
1190 DATA "omnipotent", "all-powerful"
9999 DATA "XXX"
```

EASY CHANGES

1. Add more DATA statements between lines 1000 and 9999, or replace them all with your own. Be careful not to use two or more words with very similar definitions; the program might select more than one of them as possible answers to the same question. Note that each DATA statement (or pair, for long definitions) first has the vocabulary word, then a comma, and then the definition or synonym. Be sure that you enclose the word and definition in quotes. If you add more DATA statements, you have to increase the value of D in line 300 to be at least equal to the number of words. The number of DATA statements you can have depends on how long each one is and how much user memory your computer has. Be sure to leave statement 9999 as it is – it signals that there are no more DATA statements.
2. To get something other than five choices for each question, change the value of C in line 300, and the constant in line 600. You might want only three or four choices per question.
3. If you do not want to be given a choice of how many questions are going to be asked, remove lines 260 through 280 and insert the following line:

   260 L=10:PRINT "Ten questions per set."

   This will cause 10 questions to be asked at all times. Of course, you can use some number other than 10 if you want.

MAIN ROUTINES

200-290 Prints introduction. Initializes RND function. Determines number of questions to be asked.

300-370 Reads vocabulary words and definitions into arrays. Performs housekeeping.

400-460 Selects choices for answers and determines which will be the correct one.

500-570 Determines in which format the question will be asked. Asks it.

```
650 PRINT "You are right!": QC = QC + 1
700 IF QA = 1 THEN 740
710 PRINT: PRINT "You have"; QC; "correct out of";
720 PRINT QA; "questions.": PRINT: PRINT
730 P3 = P2: P2 = P1: P1 = P(AN)
740 QA = QA + 1: IF QA <= L THEN 400
800 QA = INT(100 * QC / (QA - 1))
810 IF QA >  0 THEN 830
820 PRINT "That's really the pits!!!": GOTO 950
830 PRINT "That's a score of"; QA; "percent."
840 IF QA > 25 THEN 870
850 PRINT "At least you weren't shut out totally."
860 GOTO 950
870 IF QA > 50 THEN 900
880 PRINT "You really could use some practice."
890 GOTO 950
900 IF QA > 75 THEN 930
910 PRINT "Not bad, but practice would help."
920 GOTO 950
930 PRINT "Good!";: IF QA < 95 THEN PRINT: GOTO 950
940 PRINT "  You're almost as smart as a computer!"
950 PRINT: PRINT "Another set (hit 'RETURN') or ";
960 R$ = "": INPUT "stop (enter 'S')"; R$
970 IF R$ <> "S" AND R$ <> "s" THEN 370
980 PRINT: PRINT "See you later . . . ": PRINT
990 PRINT: PRINT: END
1000 REM -- The value of D in line 300 must be at
1010 REM    least equal to the number of different
1020 REM    words in the set.
1030 DATA "anonymous","of unknown or hidden origin"
1040 DATA "ominous", "threatening or menacing"
1050 DATA "affluent", "wealthy"
1060 DATA "apathetic","indifferent or uninterested"
1070 DATA "laconic", "terse"
1080 DATA "intrepid", "fearless or courageous"
1090 DATA "gregarious", "social or company-loving"
1100 DATA "enervated", "weak or exhausted"
1110 DATA "venerable"
1120 DATA "worthy of respect or reverence"
1130 DATA "disparate", "different and distinct"
1140 DATA "vivacious", "lively or spirited"
1150 DATA "astute", "keen in judgment"
1160 DATA "ursine", "bearlike"
1170 DATA "parsimonious", "stingy or frugal"
```

```
1180 DATA "omnicient", "all-knowing"
1190 DATA "omnipotent", "all-powerful"
9999 DATA "XXX"
```

EASY CHANGES

1. Add more DATA statements between lines 1000 and 9999, or replace them all with your own. Be careful not to use two or more words with very similar definitions; the program might select more than one of them as possible answers to the same question. Note that each DATA statement (or pair, for long definitions) first has the vocabulary word, then a comma, and then the definition or synonym. Be sure that you enclose the word and definition in quotes. If you add more DATA statements, you have to increase the value of D in line 300 to be at least equal to the number of words. The number of DATA statements you can have depends on how long each one is and how much user memory your computer has. Be sure to leave statement 9999 as it is—it signals that there are no more DATA statements.
2. To get something other than five choices for each question, change the value of C in line 300, and the constant in line 600. You might want only three or four choices per question.
3. If you do not want to be given a choice of how many questions are going to be asked, remove lines 260 through 280 and insert the following line:

   260 L=10:PRINT "Ten questions per set."

   This will cause 10 questions to be asked at all times. Of course, you can use some number other than 10 if you want.

MAIN ROUTINES

200-290 Prints introduction. Initializes RND function. Determines number of questions to be asked.

300-370 Reads vocabulary words and definitions into arrays. Performs housekeeping.

400-460 Selects choices for answers and determines which will be the correct one.

500-570 Determines in which format the question will be asked. Asks it.

600-740 Accepts answer from operator. Determines if right or wrong. Keeps score. Saves subscripts of last three correct answers.

800-990 Gives final score. Asks about doing it again.

1000-9999 DATA statements with vocabulary words and definitions.

MAIN VARIABLES

L Limit of number of questions to ask.

R Work variable to initialize RND. Also used for operator's reply to each question.

C Number of choices of answers given for each question.

D At least equal to the number of DATA statements.

D$ Array of vocabulary words.

E$ Array of definitions.

P Array for numbers of possible answers to each question.

QA Number of questions asked so far (later used to calculate percent correct).

QC Number of questions correct so far.

J,W Work, loop variables.

P1,P2,P3 Last three correct answers.

AN Subscript of correct answer in P array.

R$ Operator's reply.

Q$ Work string variable.

SUGGESTED PROJECTS

1. Modify lines 800 through 940 to display the final evaluation messages based on a finer breakdown of the percent correct. For example, show one message if 100 percent, another if 95 to 99, another if 90 to 94, etc.
2. Ask the operator's name in the introduction routine, and personalize some of the messages with his/her name.
3. Instead of just checking about the last three questions, be sure that the next question has not been asked in the last eight or ten questions. (Check lines 430 and 730.)
4. Keep track of which questions the operator misses. Then, after going through the number of questions he/she requested, repeat those that were missed.

Section 3
Game Programs

INTRODUCTION TO GAME PROGRAMS

Almost everyone likes to play games. Computer games are fun, and they provide an entertaining use of your Sorcerer. Besides providing relaxation and recreation, they have some built-in practical bonuses. They often force you to think strategically, plan ahead, or at least be orderly in your thought processes. They are also a good way to help some friends over their possible "computer phobia." We present a collection of games to fit any game-playing mood.

Maybe you desire a challenging all-skill game? Like chess or checkers, WARI involves no luck and considerable thinking. The Sorcerer will be your opponent, and a formidable one indeed.

Perhaps you're in the mood for a game with quick action and mounting excitement. GROAN is a fast-paced dice game involving mostly luck, with a dash of skill (or intuition) thrown in. The Sorcerer is ready for your challenge at any time.

JOT is a word game. You and the Sorcerer each pick secret words and then try to home in on the other's selection.

Do you like solving puzzles? If so, try DCODE. The Sorcerer will choose a secret code and then challenge you to find it. How fast can you do it?

Graphic electronic arcade games are a prevalent landmark of our times. We include two such games. RACER puts you behind the wheel of a high-speed race car. You must steer accurately to stay on course. WALLS lets you and a friend compete in a game of cut and thrust. Each of you must avoid crossing the path laid by the other, and by yourself!

DECODE

PURPOSE

DCODE is really more of a puzzle than a game, although you can still compete with your friends to see who can solve the puzzles the fastest. Each time you play, you are presented with a new puzzle to solve.

The object is to figure out the computer's secret code in as few guesses as possible. The program gives you information about the accuracy of each of your guesses. By carefully selecting your guesses to make use of the information you have, you can determine what the secret code must be in a surprisingly small number of guesses. Five or six are usually enough.

The first few times you try, you will probably require quite a few more guesses than that, but with practice, you'll discover that you can learn a lot more from each guess than you originally thought.

HOW TO USE IT

The program starts off by displaying a brief introduction, and then selecting a secret code for you to discover. The code is a 4-digit number that uses only the digits 1 through 6. For example, your Sorcerer might pick 6135 or 2242 as a secret code.

Your object is to guess the code in the fewest possible guesses. After each of your guesses, the program tells you a "black" and a "white" number. The black number indicates the number of digits in your guess that were correct—that is, the digit was correct *and* in the correct position. So, if the secret code is 6153 and your guess is 4143, you will be told that black is 2 (because the 1 and the 3 are correct). Of course, you aren't told *which* digits are correct. That is for you to figure out by making use of the information you get from other guesses.

Each of the white numbers indicates a digit in your guess that was correct, but which was in the wrong position. For example, if the secret code is 6153 and your guess is 1434, you will be told that white is 2. The 1 and 3 are correct, but in wrong positions.

The white number is determined by ignoring any digits that accounted for a black numbers. A single position in the secret code or guess can only account for one black or white number. These facts become significant when the secret code and/or your guess have duplicate digits. For example, if the code is 1234 and your guess is 4444, there is only one black, and no whites. If the code is 2244 and your guess is 4122, there are no blacks and three whites. This may sound a little tricky, but you will quickly get the hang of it.

At any time during the game, you can ask for a summary by entering **S** instead of a guess. This causes the program to clear the screen and display each guess (with the corresponding result) that has occurred so far.

Also, if you get tired of trying and want to give up, you can enter Q (for "quit") to end your misery and find out the answer. Otherwise, you continue guessing until you get the code right (four black, zero white), or until you have used up the maximum of 12 guesses.

SAMPLE RUN

```
* * * DECODE * * *

The object of the game is to deduce a 4 position
code number, made up of the digits 1 thru 6 only.

The machine will indicate the number of digits
both correctly guessed, and guessed at the right
position, and label that value BLACK.

WHITE indicates that a digit is correct, but in
the wrong position.
```

```
Input a random value, please? 1459

The secret code has been selected.

Enter 'S' for a summary or 'Q' to quit at any
time.

Your guess # 1 (or 'S' or 'Q') -- ? 1122

Guess # 1 -- Black = 2       White = Ø

Your guess # 2 (or 'S' or 'Q') -- ? 2233

Guess # 2 -- Black = Ø       White = 1

Your guess # 3 (or 'S' or 'Q') -- ? 1332

Guess # 3 -- Black = 2       White = Ø

Your guess # 4 (or 'S' or 'Q') -- ? 1562

Guess # 4 -- Black = 2       White = 2

Your guess # 5 (or 'S' or 'Q') -- ? S

* * * SUMMARY * * *

Guess #  Guess   Black    White
-------  -----   -----    -----
 1       1122     2        Ø
 2       2233     Ø        1
 3       1332     2        Ø
 4       1562     2        2

Your guess # 5 (or 'S' or 'Q') -- ? 1652

Guess # 5 -- Black = 4        White = Ø

That's it -- 1652!!

You got it in 5 guesses -- pretty good!

Another game (hit 'RETURN') or
stop (enter 'S')? S
```

PROGRAM LISTING

```
1ØØ REM <DCODE>
11Ø REM Decoding game
12Ø REM For the Sorcerer (TM Exidy Inc.)
13Ø REM Copyright 1979
```

```
140 REM By Kevin McCabe, Tom Russ & Phil Feldman
150 CLEAR 1000: DIG = 6: PO = 4: MG = 12
160 DIM G$(MG), G(PO), C(PO), BL(MG), WH(MG)
170 GOSUB 1200
180 GOSUB 300: GOSUB 400
200 PRINT "Your guess #"; GN; "(or ´S´ or ´Q´)";
205 INPUT " -- "; R$
210 IF LEFT$(R$,1)="S" OR LEFT$(R$,1)="s" THEN 500
220 IF LEFT$(R$,1)="Q" OR LEFT$(R$,1)="q" THEN 600
230 GOSUB 700: GUSUB 800: GOSUB 1000
240 IF BL(GN) = PO THEN 2000
250 G$(GN) = R$: GN = GN + 1: IF GN > MG THEN 2200
260 GOTO 200
300 GN = 1: C$ = "": RETURN
400 FOR J = 1 TO PO: R = INT(DIG * RND(1)) + 1
410 C$ = C$ + RIGHT$(STR$(R),1): NEXT J: PRINT
420 PRINT "The secret code has been selected."
430 PRINT:PRINT:PRINT "Enter ´S´ for a summary ";
440 PRINT "or ´Q´ to quit at any time.": PRINT
450 RETURN
500 IF GN > 1 THEN 520
510 PRINT "No guesses yet!": GOTO 200
520 PRINT CHR$(12), "* * *  SUMMARY  * * *": PRINT
530 PRINT "Guess #", "Guess", "Black", "White"
540 PRINT "-------","-------","-------","-------"
550 FOR J = 1 TO GN - 1
560 PRINT J, G$(J), BL(J), WH(J)
570 IF GN < 12 THEN PRINT
580 NEXT J: GOTO 200
600 PRINT: PRINT "Can´t take it, huh?  Well, the ";
610 PRINT "secret code was . . .": PRINT
620 FOR J = 1 TO 500: NEXT J: PRINT TAB(20); C$
630 PRINT: GOTO 2100
700 IF LEN(R$) = PO THEN 720
710 PRINT: PRINT "ILLEGAL!  Try again!": GOTO 200
720 FOR J = 1 TO PO: R = VAL(MID$(R$,J,1))
730 IF R < 1 OR R > DIG THEN 710
740 NEXT J: RETURN
800 B = 0: W = 0: FOR J = 1 TO PO
810 G(J) = VAL(MID$(R$,J,1))
820 C(J) = VAL(MID$(C$,J,1))
830 IF C(J)=G(J) THEN B = B+1: G(J) = 0: C(J) = 0
840 NEXT J: FOR J = 1 TO PO: IF C(J) = 0 THEN 890
850 H = 0: FOR K = 1 TO PO: IF C(J) = 0 THEN 880
860 IF C(J) <> G(K) THEN 880
```

```
870 H = 1: G(K) = 0: C(J) = 0
880 NEXT K: W = W + H
890 NEXT J: RETURN
1000 BL(GN) = B: WH(GN) = W: PRINT
1010 PRINT "Guess #"; GN; "-- Black ="; B,
1020 PRINT "White ="; W: PRINT: RETURN
1200 PRINT CHR$(12), "* * *  DECODE  * * *"
1210 PRINT: PRINT: PRINT "The object of the game";
1220 PRINT " is to deduce a"; PO; "position code"
1230 PRINT "number, made up of the digits 1 thru";
1240 PRINT DIG; "only.": PRINT: PRINT
1250 PRINT "The machine will indicate the number";
1260 PRINT " of digits both correctly"
1270 PRINT "guessed, and guessed at the right ";
1280 PRINT "position, and label that valueBLACK."
1290 PRINT: PRINT "WHITE indicates that a digit ";
1300 PRINT "is correct, but in the wrong"
1310 PRINT "position.": PRINT: PRINT
1320 INPUT "Input a random value, please"; R$
1330 R = RND(-ABS(VAL(R$))): PRINT: PRINT: R$ = ""
1340 RETURN
2000 PRINT: PRINT "That's it -- "; C$; "!!": PRINT
2010 PRINT "You got it in"; GN; "guesses -- ";
2020 IF GN>8 THEN PRINT "pretty slow!": GOTO 2100
2030 IF GN>6 THEN PRINT "not bad!": GOTO 2100
2040 IF GN>4 THEN PRINT "pretty good!": GOTO 2100
2050 PRINT "Outstanding!!!"
2100 PRINT:PRINT"Another game (hit 'RETURN') or ";
2110 INPUT "stop (enter 'S')"; R$
2120 IF R$ = "S" OR R$ = "s" THEN END
2130 GOTO 180
2200 PRINT:PRINT:PRINT"Sorry, that's your limit";
2210 PRINT " of"; MG; "guesses.": PRINT
2220 PRINT "The secret code was "; C$; "."
2230 GOTO 2100
```

EASY CHANGES

1. Modify line 150 to change the complexity of the code and/or the number of guesses you are allowed. For example, the following line would allow 15 guesses at a five-position code using the digits 1 through 8:

   150 CLEAR 1000:DIG=8:PO=5:MG=15

   The introduction will automatically reflect the new values for

DIG and PO. Be sure that DIG is not set greater than 9.

2. To change the program so it will automatically display the "Summary" information after each guess, replace line 260 with:

```
260 GOTO 500
```

MAIN ROUTINES

150-180 Initializes variables. Displays introduction. Chooses secret code.

200-230 Gets a guess from operator. Analyzes reply. Displays result.

240 Determines if operator guessed correctly.

250-260 Saves guess. Adds one to guess counter. Determines if limit on number of guesses was exceeded.

300 Subroutine to initialize variables.

400-450 Subroutine to choose secret code and inform operator.

500-580 Routine to display summary of guesses so far.

600-630 Routine to slowly display secret code when operator quits.

700-740 Subroutine to determine if operator's guess was legal.

800-890 Subroutine to determine number of black and white responses for the guess.

1000-1020 Subroutine to display number of black and white responses for the guess.

1200-1340 Subroutine to display title and introduction.

2000-2130 Routine to analyze operator's performance after correct answer is guessed and ask about playing again.

2200-2230 Routine to display secret code after operator exceeds limit of number of guesses.

MAIN VARIABLES

DIG Number of possible digits in each position of the code (i.e., a digit from 1 to DIG).

PO Number of positions in the code.

MG Limit on number of guesses that can be made.

G$ Array in which guesses are saved.

G,C	Work arrays in which each guess is analyzed.
BL,WH	Arrays in which the number of black and white responses is saved for each guess.
R,H	Work variables.
GN	Counter of the number of guesses made.
R$	Reply by the operator.
C$	Secret code chosen by the program.
J,K	Loop variables.
B,W	Number of black and white responses for this guess.

SUGGESTED PROJECTS

1. Change the analysis at the end of the game to take into account the difficulty of the code as well as the number of guesses it took to figure out the code. A 4-position code using the digits 1 through 6 has 1296 possibilities, but a 5-position code using 1 through 8 has 32768 possibilities. Change lines 2020 through 2050 to determine the message to be displayed based on the number of possibilities in the code as well as on GN.
2. At the beginning of the game, give the operator the option of deciding the complexity of the code. Ask for the number of positions and the number of digits. Make sure only "reasonable" numbers are used—do not try to create a code with zero positions, for example. Another approach is to ask the operator if he/she wants to play the easy, intermediate, or advanced version. Then set the values of DIG and PO accordingly. Suggestions are:

Easy:	DIG=3 and PO=3
Intermediate:	DIG=6 and PO=4
Advanced:	DIG=8 and PO=5

GROAN

PURPOSE

Do you like the thrills of fast-paced dice games? If so, GROAN is right up your alley. It is a two-person game with the computer playing directly against you. There is a considerable amount of luck involved. However, the skill of deciding when to pass the dice to your opponent also figures prominently.

The Sorcerer will roll the dice for both players, but don't worry—it will not cheat! (We wouldn't think of stooping to such depths.)

Why is the game called GROAN? You will know the answer to this question very shortly after playing the game.

HOW TO USE IT

The game uses two dice. They are just like regular six-sided dice except for one thing. The die face where the "1" would normally be has a picture of a frowning face instead. The other five faces of each die have the usual numbers two through six on them.

The object is to be the first player to achieve a score agreed upon before the start of the game. Players alternate taking turns. A turn consists of a series of dice rolls (at least one roll, possibly several) subject to the following rules.

As long as no frown appears on either die, the roller builds a running score for this current series of rolls. After each roll with no frown, he has the choice of rolling again or passing the dice to his opponent. If he passes the dice, his score achieved on the current series is added to any previous total he may have had.

But if he rolls and a frown appears, he will be groaning. A frown on only one die cancels any score achieved for the current series of rolls. Any previous score is retained in this case. However, if he rolls a double frown, his entire previous total is wiped out as well as his current total. Thus, he reverts back to a total score of zero – true despair.

The program begins by getting a random value, and then asking what the winning score should be. Values between 50 and 100 tend to produce the best games, but any positive value is acceptable. Next, a simulated coin toss randomly decides who will get the first roll.

Each dice roll is portrayed with a short graphics display. Before each roll, the Sorcerer indicates whose roll is coming up.

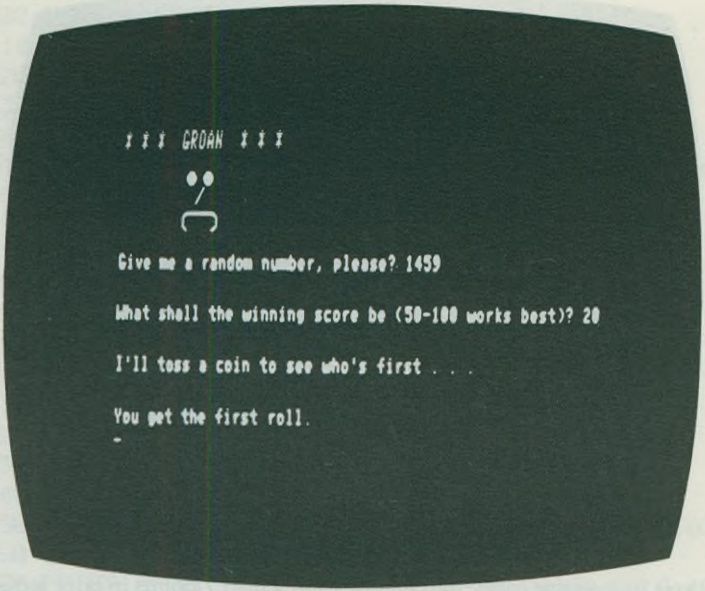
Each roll is followed by a display of the scoreboard. This scoreboard gives all relevant information: score needed to win, both players' scores before the current series of rolls, and the total score for the current series.

If a frown should appear on a die, the scoreboard will indicate the current running total as zero. In addition, the previous total will become zero in the case of the dreaded double frown. In either case, the dice will be passed automatically to the next player.

If a scoring roll results, the roller must decide whether to roll again or to pass the dice. The program has a built-in strategy to decide this for the Sorcerer. For you, the question will be asked after the scoreboard is displayed. The two legal replies are **P** and **R**. The **R** means that you wish to roll again. The **P** means that you choose to pass the dice to the Sorcerer. If you should score enough to win, you must still pass the dice to add the current series to your previous total.

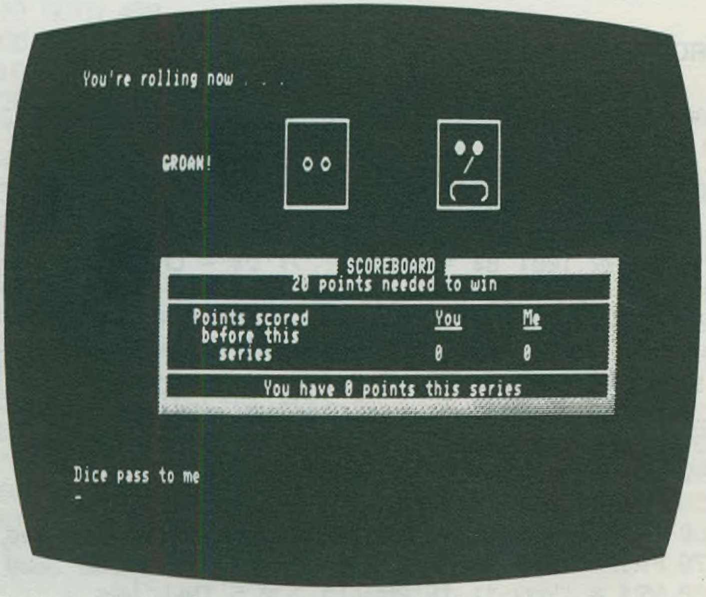
The first player to pass the dice with a score greater than or equal to the winning score is the victor. This will surely cause his opponent to GROAN. The Sorcerer will acknowledge the winner.

SAMPLE RUN



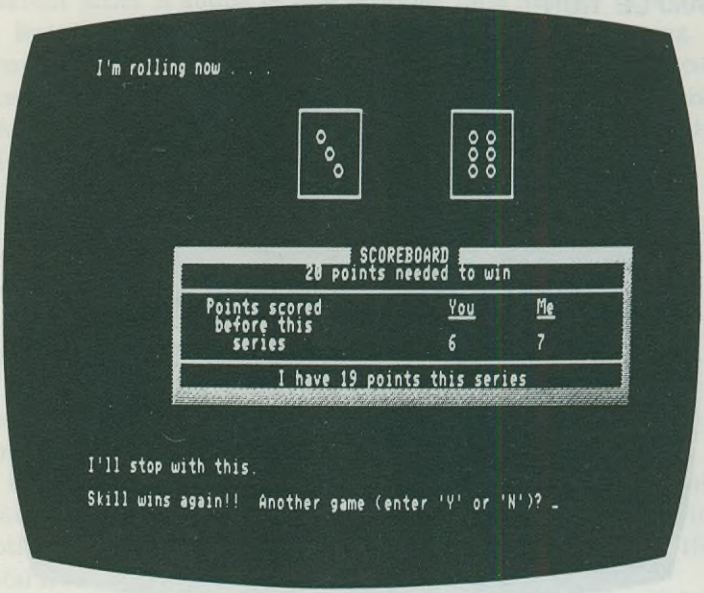
```
* * * GROAN * * *

Give me a random number, please? 1459

What shall the winning score be (50-100 works best)? 20

I'll toss a coin to see who's first . . .

You get the first roll.
-
```

The user has challenged the Sorcerer to a twenty-point game, and has won the toss for the first roll.



```
You're rolling now . . .

          GROAN!

               SCOREBOARD
          20 points needed to win

      Points scored          You     Me
       before this
         series              0       0

          You have 0 points this series

Dice pass to me
-
```

The first roll, though, results in a two and a "groan," so no points are scored. The dice pass to the Sorcerer.



```
I'm rolling now . . .

                        SCOREBOARD
                  20 points needed to win
     Points scored                 You       Me
      before this
        series                     6         7
                 I have 19 points this series

I'll stop with this.

Skill wins again!!  Another game (enter 'Y' or 'N')? _
```

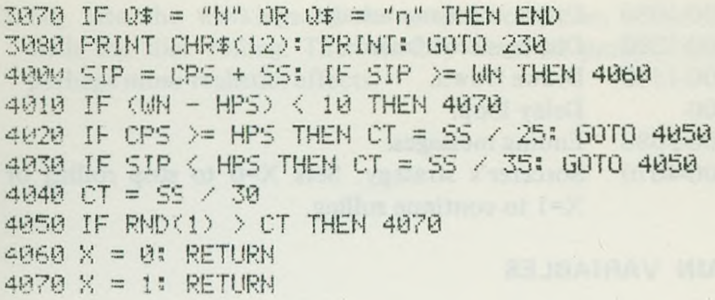
Later in the same game, the computer has scored 7 points in prior series and 19 in the current one. Since the total exceeds 20, the Sorcerer wins again.

PROGRAM LISTING

```
100 REM <GROAN>
110 REM Groan -- a dice game
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 CLEAR 100: B$ = CHR$(177): D$ = CHR$(151)
160 U$ = CHR$(137): TL = 10: TR = 61
200 PRINT CHR$(12); "* * *  GROAN  * * *": PRINT
205 C = -3578: GOSUB 1500: PRINT: PRINT: PRINT
210 PRINT: PRINT: PRINT "Give me a random number,";
220 INPUT " please"; Q$: Q = RND(-ABS(VAL(Q$)))
230 PRINT: PRINT: PRINT "What shall the winning ";
240 INPUT "score be (50-100 works best)"; WN
250 WN = INT(WN): IF WN < 1 THEN 230
260 PRINT: PRINT: PRINT "I'll toss a coin to see ";
270 PRINT "who's first . . .": PRINT: GOSUB 2000
280 NR$ = "You ": IF RND(1) > 0.5 THEN NR$ = "I "
285 PRINT: PRINT NR$; "get the first roll.": SS=0
```

```
290 GOSUB 2000: IF NR$ = "I " THEN 400
300 PRINT CHR$(12):PRINT "You're rolling now . . ."
310 NR$ = "You ": GOSUB 500
320 SS = SS + R1 + R2: IF F > 0 THEN SS = 0
330 ON F GOSUB 1000, 1020: IF F>1 THEN HPS = 0
340 GOSUB 1100: PRINT: IF F < 1 THEN 360
350 PRINT "Dice pass to me.": GOSUB 2000:GOTO 400
360 PRINT "Do you want to pass the dice (enter ";
370 INPUT "'P') or roll ('R')"; Q$
375 IF Q$ = "R" OR Q$ = "r" THEN 300
380 IF Q$ <> "P" AND Q$ <> "p" THEN 360
385 HPS = HPS + SS: IF HPS >= WN THEN 3000
390 SS = 0: PRINT: GOTO 350
400 PRINT CHR$(12): PRINT "I'm rolling now . . ."
410 NR$ = "I ": GOSUB 500
420 SS = SS + R1 + R2: IF F > 0 THEN SS = 0
430 ON F GOSUB 1000, 1020: IF F>1 THEN CPS = 0
440 GOSUB 1100: PRINT: IF F < 1 THEN 460
450 PRINT"Dice pass to you.":GOSUB2000:SS=0:GOTO300
460 GOSUB 4000: IF X = 0 THEN 480
470 PRINT "I'll roll again.": GOSUB 2000: GOTO 400
480 PRINT "I'll stop with this.  ";
485 CPS = CPS + SS: IF CPS >= WN THEN 3000
490 GOTO 450
500 GOSUB 600
510 FOR J = 1 TO 5
520 F = 0: C = -3368: R1 = INT(6*RND(1) + 1)
530 IF R1 > 6 THEN R1 = 6
540 R = R1: GOSUB 700: C = -3351
550 R2 = INT(6*RND(1) + 1): IF R2 > 6 THEN R2 = 6
560 R = R2: GOSUB 700: IF R1>1 AND R2>1 THEN 590
570 IF R1 = 1 THEN F = 1
580 IF R2 = 1 THEN F = F + 1
590 NEXT J: RETURN
600 C = -3368: GOSUB 650: C = -3351: GOSUB 650
610 RETURN
650 FOR K=C TO C+6: POKE K, 137: NEXT K
660 FOR K=C-58 TO C-314 STEP -64: POKE K,135:NEXT K
670 FOR K=C-64 TO C-320 STEP -64: POKE K,128:NEXT K
680 FOR K=C-384 TO C-378: POKE K, 176: NEXT K
690 RETURN
700 FOR K=C-63 TO C-319 STEP -64: FOR L=0 TO 4
710 POKE K+L, 32: NEXT L: NEXT K
720 Q=136: ON R GOTO 730,740,750,760,770,780
```

```
730 GOSUB 1500: GOTO 790
740 POKE C-188, Q: POKE C-190, Q: GOTO 790
750 POKE C-254,Q: POKE C-124,Q: POKE C-189,Q
755 GOTO 790
760 POKE C-254,Q: POKE C-252,Q: POKE C-126,Q
765 POKE C-124, Q: GOTO 790
770 POKE C-189, Q: GOTO 760
780 POKE C-188,Q: POKE C-190,Q: GOTO 760
790 RETURN
1000 PRINT: PRINT: PRINT: PRINT
1010 PRINT TAB(TL); "GROAN!": RETURN
1020 PRINT: PRINT: PRINT: PRINT
1030 PRINT TAB(TL); "DESPAIR!!": RETURN
1100 PRINT CHR$(17);: FOR J = 1 TO 12: PRINT:NEXT J
1110 PRINT TAB(TL);: FOR J = TL + 1 TO TL+20
1120 PRINT B$;: NEXT J: PRINT " SCOREBOARD ";
1130 FOR J = TL+33 TO TR+1: PRINT B$;: NEXT J
1140 PRINT: PRINT TAB(TL); B$; SPC(13); WN;
1150 PRINT "points needed to win"; TAB(TR); B$
1160 PRINT TAB(TL); B$;: FOR J = TL+2 TO TR
1170 PRINT D$;: NEXT J: PRINT B$: PRINT TAB(TL);B$;
1180 PRINT "   Points scored"; TAB(40); " You";
1190 PRINT TAB(50); " Me"; TAB(TR); B$
1200 PRINT TAB(TL); B$; "    before this"; TAB(41);
1210 PRINT U$;U$;U$; TAB(51); U$;U$; TAB(TR); B$
1220 PRINT TAB(TL); B$; "       series"; TAB(40);
1230 PRINT HPS; TAB(50); CPS; TAB(TR); B$
1240 PRINT TAB(TL); B$;: FOR J = TL+2 TO TR
1250 PRINT D$;: NEXT J: PRINT TAB(TL); B$
1260 PRINT TAB(TL); B$; SPC(11); NR$; "have"; SS;
1270 PRINT "points this series"; TAB(TR); B$
1280 PRINT TAB(TL);: FOR J = TL+1 TO TR+1
1290 PRINT B$;: NEXT J: PRINT: PRINT: PRINT: RETURN
1500 POKE C-254, 132: POKE C-252, 132
1510 POKE C-189, 171: POKE C-127, 143
1520 POKE C-123, 144: FOR L = C-126 TO C-124
1530 POKE L, 151: NEXT L
1540 POKE C-63, 154: POKE C-59, 155: RETURN
2000 FOR K = 1 TO 1500: NEXT K: RETURN
3000 PRINT: IF CPS < WN THEN 3030
3020 PRINT: PRINT "Skill wins again!!  ";
3030 IF HPS < WN THEN 3050
3040 PRINT: PRINT "You win -- luck, just luck!  ";
3050 PRINT "Another game (enter 'Y' or 'N')";
3060 INPUT Q$: HPS = 0: CPS = 0
```



```
3070 IF Q$ = "N" OR Q$ = "n" THEN END
3080 PRINT CHR$(12): PRINT: GOTO 230
4000 SIP = CPS + SS: IF SIP >= WN THEN 4060
4010 IF (WN - HPS) < 10 THEN 4070
4020 IF CPS >= HPS THEN CT = SS / 25: GOTO 4050
4030 IF SIP < HPS THEN CT = SS / 35: GOTO 4050
4040 CT = SS / 30
4050 IF RND(1) > CT THEN 4070
4060 X = 0: RETURN
4070 X = 1: RETURN
```

EASY CHANGES

1. If you wish to set the program for a fixed value of the winning score, it can be done by changing line 230. Simply set WN to the winning score desired. For example,

   230 WN=100:GOTO 260

   would make the winning score 100.
2. The rotating dice graphics display before each roll can be eliminated by

   510 FOR J=1 TO 1

   This has the effect of speeding up the game by showing each dice roll immediately.
3. After you play the game a few times, you may wish to change the delay constant in line 2000. It controls the "pacing" of the game; i.e., the time delays between various messages, etc. To speed up the game try

   2000 FOR K=1 TO 750:NEXT K:RETURN

   Of course, if desired, the constant can be set to larger values to slow down the game.

MAIN ROUTINES

150-160	Initializes constants.
200-290	Initial display. Gets winning score.
300-390	Human rolls.
400-490	Sorcerer rolls.
500-590	Determines dice roll. Drives its display.
600-690	Draws die outline.
700-790	Draws die face.

1000-1030	Displays groan messages.
1100-1290	Displays scoreboard.
1500-1540	Draws frown.
2000	Delay loop.
3000-3080	Ending messages.
4000-4070	Sorcerer's strategy. Sets X=0 to stop rolling or X=1 to continue rolling.

MAIN VARIABLES

WN	Amount needed to win.
HPS	Previous score of human.
CPS	Previous score of Sorcerer.
SS	Score of current series of rolls.
X	Sorcerer strategy flag (0=stop rolling; 1=roll again).
CT	Cutoff threshold used in Sorcerer's built-in strategy.
SIP	Score Sorcerer would have if it passed the dice.
B$,D$,U$	Characters for border, divider, underline.
Q,Q$	Work variable, work string variable.
J,K,L	Loop indices.
NR$	String of name of current roller.
R1,R2	Outcome of roll for die 1, die 2.
R	Outcome of a die roll.
F	Result of roll (0=no frown; 1=one frown; 2= double frown).
C	Poke address of a die.
TL,TR	TAB arguments.

SUGGESTED PROJECTS

1. The Sorcerer's built-in strategy is contained from line 4000 on. Remember, after a no-frown roll, the Sorcerer must decide whether or not to continue rolling. See if you can improve on the current strategy. You may use, but not modify, the variables CPS, HPS, SS, and WN. The variable X must be set before returning. Set X=0 to pass the dice or X=1 to roll again.
2. Ask the operator for his/her name. Then personalize the messages and scoreboard.

3. Dig into the workings of the graphics routines connected with the dice rolling. Then modify them to produce new, perhaps more realistic, effects.

JOT

PURPOSE

JOT is a two-player word game involving considerable mental deduction. The Sorcerer will play against you. But be careful! You will find your computer to be quite a formidable opponent.

The rules of JOT are fairly simple. The game is played entirely with three-letter words. All letters of each word must be distinct—no repeats. (See the section on Easy Changes for further criteria used in defining legal words.)

To begin the game, each player chooses a legal secret word. The remainder of the game involves trying to be the first player to deduce the other's secret word.

The players take turns making guesses at their opponent's word. After each guess, the player is told how many letters (or "hits") his guess had in common with his opponent's secret word. The position of the letters in the word does not matter. For example, if the secret word were "own," a guess of "who" would have 2 hits. The winner is the first person to correctly guess his opponent's secret word.

HOW TO USE IT

The program begins with some introductory messages while asking you to select your secret word from the list of "legal" words. It then asks whether or not you wish to make the first guess. This is followed by you and the Sorcerer alternating guesses at each other's secret word.

After the Sorcerer guesses, it will immediately ask you how it did. Possible replies are **0, 1, 2, 3,** or **R**. The response of **R** (for right) means the Sorcerer has just guessed your word correctly – a truly humbling experience. The numerical replies indicate that the word guessed by the Sorcerer had that number of hits in your secret word. A response of 3 means that all the letters were correct, but they need to be rearranged to form the actual secret word (e.g. a guess of "EAT" with the secret word being "TEA").

After learning how it did, the computer will take some time to process its new information. If this time is not trivial, the Sorcerer will display the message "I'm thinking" so you do not suspect it of idle daydreaming. If it finds an inconsistency in its information, it will ask you for your secret word and then analyze what went wrong.

When it is your turn to guess, there are two special replies you can make. These are the single letters **S** or **Q**. The **S**, for summary, will display a table of all previous guesses and corresponding hits. This is useful as a concise look at all available information. It will then prompt you again for your next guess. The **Q**, for quit, will simply terminate the game.

When not making one of these special replies, you will input a guess at the Sorcerer's secret word. This will be, of course, a 3-letter, *uppercase* word. If the word used is not legal, the computer will so inform you. After a legal guess, you will be told how many hits your guess had. If you correctly guess the Sorcerer's word, you will be duly congratulated. The computer will then ask you for your secret word and verify that all is on the "up and up."

SAMPLE RUN

```
* * *  JOT  * * *


Give me a random value, please? 1459

Thanks.  Just a second . . .

OK, now let's each think of a secret 3 letter
word, with no repeated letters.

The 'SHIFT LOCK' key must be DOWN!

Do you need a list of legal words (enter 'Y'
or 'N')? N
```

```
Now, let me think . . . Got it!

Do you want to go first (enter 'Y' or 'N')? Y

Your guess (or enter 'S' or 'Q')? GUN
You had 1 hits on that one.

My guess is MIX.
How did I do (enter 'Ø', '1', '2', '3',
or 'R')? Ø
I'm thinking . . .

Your guess (or enter 'S' or 'Q')? FUN
You had Ø hits on that one.

My guess is RAN.
      :
   (later in the same game)
      :
Your guess (or enter 'S' or 'Q')? S

                SUMMARY
 YOUR GUESSES                  MY GUESSES
Word      Hits    Number    Word     Hits
-----------------------------------------
GUN        1        1       MIX       Ø
FUN        Ø        2       RAN       1
GOT        1        3       WAG       Ø
GAS        2        4       TON       3

Your guess (or enter 'S' or 'Q')? GAP
You had 2 hits on that one.

My guess is NOT.
How did I do (enter 'Ø', '1', '2', '3',
or 'R')? R

Gotcha that time!! My word was JAG.

How about another game (enter 'Y' or 'N')? N
```

PROGRAM LISTING

```
100 REM <JOT>
110 REM Jot game
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 CLEAR 2500
160 MG = 25: MW = 500
170 DIM WL$(MW), HG$(MG), CG$(MG), HH(MG), CH(MG)
```

```
200 PRINT CHR$(12);"* * *  J O T  * * *"
220 PRINT: PRINT: PRINT "Give me a random value, ";
230 INPUT "please"; R$: R = RND(-ABS(VAL(R$)))
240 PRINT: PRINT "Thanks.  Just a second . . ."
250 W = 0: RESTORE: GOSUB 3000
255 HNG = 0: CNG = 0: HPSW = NW: PRINT
260 PRINT "OK, now let's each think of a secret ";
270 PRINT "3 letter word, with"
280 PRINT "no repeated letters.":PRINT:PRINT "The ";
285 PRINT "'SHIFT LOCK' must be DOWN!": PRINT
290 PRINT "Do you need a list of legal words (";
300 INPUT "enter 'Y' or 'N')"; R$: PRINT
310 IF R$ = "Y" THEN GOSUB 3500
315 IF R$ = "y" OR R$ = "n" THEN GOTO 285
320 PRINT "Now, let me think . . .";: GOSUB 2200
330 CSW$ = WL$(Q): PRINT "Got it!"
340 PRINT: PRINT "Do you want to go first (enter ";
350 INPUT "'Y' or 'N')"; R$: R$ = LEFT$(R$,1)
360 IF R$ = "Y"  THEN 500
370 IF R$ = "N" THEN 600
380 PRINT "Check that the 'SHIFT LOCK' key is ";
390 PRINT "down, and say again?": GOTO 340
500 PRINT: PRINT: PRINT "Your guess (or enter 'S'";
510 INPUT " or 'Q')"; R$: IF R$ = "Q" THEN 1200
520 IF R$ = "S" THEN GOSUB 1000: GOTO 500
530 IF R$ <> CSW$ THEN 550
540 HNG=HNG+1: HG$(HNG)=R$: HH(HNG)=9: GOTO 3400
550 GOSUB 1800: IF F <> 0 THEN 570
560 PRINT "ILLEGAL WORD -- try again!": GOTO 500
570 Q$ = CSW$: GOSUB 2600: Q$ = R$: GOSUB 1500
580 HNG = HNG + 1: HH(HNG) = Q: HG$(HNG) = R$
585 PRINT "You had"; Q; "hits on that one."
590 IF HNG = MG THEN 3600
600 Q$ = WL$(HPSW): CNG = CNG + 1: CG$(CNG) = Q$
610 PRINT: PRINT "My guess is "; Q$; "."
620 PRINT "How did I do (enter '0', '1', '2', '3'";
630 INPUT ", or 'R')"; R$: R$ = LEFT$(R$,1)
640 IF R$ = "R" THEN  CH(CNG) = 9: GOTO 3200
650 P = INT(VAL(R$)): IF P>=0 AND P<4 THEN 670
660 PRINT "Come on, try it again. . .": GOTO 610
670 IF P = 0 AND R$ <> "0" THEN 660
675 IF HPSW > 100 THEN PRINT "I'm thinking . . ."
680 CH(CNG) = P: GOSUB 800: GOTO 500
800 Q$ = CG$(CNG): H = CH(CNG): J = 0: GOSUB 2600
810 HPSW = HPSW - 1: IF HPSW < 1 THEN 900
```

```
820 J = J + 1: IF J > HPSW THEN RETURN
830 Q$ = WL$(J): GOSUB 1500: IF Q = H THEN 820
840 Q = J: P = HPSW: GOSUB 2400: HPSW = HPSW - 1
850 IF HPSW < 1 THEN 900
860 IF HPSW >= J THEN 830
870 RETURN
900 PRINT: PRINT: PRINT "Something's wrong!!"
910 INPUT "What was your secret word"; R$
920 GOSUB 1800: IF F <> 0 THEN 940
930 PRINT"ILLEGAL--I never had a chance!!":GOTO1200
940 PRINT: PRINT: PRINT "You gave a bad answer ";
950 PRINT "somewhere--check the summary!"
960 GOSUB 1000: GOTO 1200
1000 PRINT: PRINT: Q = HNG: IF CNG > Q THEN Q = CNG
1010 IF Q = 0 THEN PRINT "No guesses yet!": RETURN
1020 PRINT:PRINT:PRINT TAB(23);"S U M M A R Y"
1030 PRINT " YOUR GUESSES"; TAB(42); "MY GUESSES"
1040 PRINT "Word"; TAB(10); "Hits"; TAB(25);
1050 PRINT "Number"; TAB(40); "Word";THB(50);"Hits"
1060 FOR J=1 TO 55: PRINT "-";: NEXT J: PRINT
1070 FOR J = 1 TO Q: IF J > HNG THEN 1100
1090 PRINT HG$(J); TAB(10); HH(J);
1100 PRINT TAB(25); J;: IF J > CNG THEN 1120
1110 PRINT TAB(40); CG$(J); TAB(50); CH(J);
1120 PRINT: NEXT J: PRINT: RETURN
1200 PRINT: PRINT: PRINT "How about another game ";
1210 INPUT "(enter 'Y' or 'N')"; R$
1220 IF R$ = "Y" THEN 240
1230 IF R$ = "N" THEN END
1240 PRINT "Come on, 'Y' or 'N' please!": GOTO 1200
1500 P$ = LEFT$(Q$,1): Q = 0: GOSUB 1530
1510 P$ = MID$(Q$,2,1): GOSUB 1530
1520 P$ = RIGHT$(Q$,1)
1530 IF P$ = M1$ OR P$ = M2$ OR P$ = M3$ THEN Q=Q+1
1540 RETURN
1800 J = 0: F = 0: IF LEN(R$) <> 3 THEN RETURN
1810 J = J + 1: IF J > NW THEN RETURN
1820 Q$ = WL$(J): IF Q$ <> R$ THEN 1810
1830 F = 1: RETURN
2200 FOR J = 1 TO NW
2210 P = INT(NW * RND(1)) + 1: IF P > NW THEN 2210
2220 Q = INT(NW * RND(1)) + 1: IF Q > NW THEN 2220
2230 GOSUB 2400: NEXT J: RETURN
2400 P$ = WL$(P): WL$(P) = WL$(Q): WL$(Q) = P$
2410 RETURN
```

```
2600 M1$ = LEFT$(Q$,1): M2$ = MID$(Q$,2,1)
2610 M3$ = RIGHT$(Q$,1): RETURN
3000 READ W$: IF W$ = "XXX" THEN NW = W: RETURN
3010 W = W + 1: WL$(W) = W$: IF W <= MW THEN 3000
3020 PRINT "Too many words -- RUN ABORTED!": END
3200 PRINT: PRINT "Gotcha that time!!  ";
3210 PRINT "My word was "; CSW$; ".": GOTO 1200
3400 PRINT: PRINT "CONGRATULATIONS!  That's it!"
3410 PRINT: INPUT "What was your word"; R$
3420 GOSUB 1800: J = 1: IF F <> 0 THEN 3440
3430 PRINT "ILLEGAL -- I had no chance!!":GOTO 1200
3440 Q$ = WL$(J): IF Q$ <> R$ THEN 3460
3450 PRINT: PRINT "Nice word!": PRINT: GOTO 1200
3460 J = J + 1: IF J <= HPSW THEN 3440
3470 PRINT: PRINT "You made an error somewhere --";
3480 PRINT " check the summary!"
3490 GOSUB 1000: GOTO 1200
3500 PRINT:PRINT"Use the 'RUN/STOP' or 'ESC' to ";
3510 PRINT "halt in mid-list.": PRINT
3520 FOR J = 1 TO NW
3530 IF J/11 = INT(J/11) THEN PRINT: PRINT
3540 PRINT WL$(J); "  ";: NEXT J: PRINT: PRINT
3550 RETURN
3600 PRINT "That's all the memory space I have."
3610 PRINT: PRINT: PRINT "My word was "; CSW$; "."
3620 GOTO 1200
5000 DATA "ACE","ACT","ADE","ADO","ADS","AFT","AGE"
5010 DATA "AGO","AID","AIL","AIM","AIR","ALE","ALP"
5020 DATA "AND","ANT","ANY","APE","APT","ARC","ARE"
5030 DATA "ARK","ARM","ART","ASH","ASK","ASP","ATE"
5040 DATA "AWE","AWL","AXE","AYE","BAD","BAG","BAN"
5050 DATA "BAR","BAT","BAY","BED","BEG","BET","BID"
5060 DATA "BIG","BIN","BIT","BOA","BOG","BOW","BOX"
5070 DATA "BOY","BUD","BUG","BUM","BUN","BUS","BUT"
5080 DATA "BUY","BYE","CHB","CAD","CAM","CAN","CAP"
5090 DATA "CAR","CAT","COB","COD","COG","CON","COP"
5100 DATA "COT","COW","CUY","CRY","CUB","CUD","CUE"
5110 DATA "CUP","CUR","CUT","DAB","DAM","DAY","DEN"
5120 DATA "DEW","DIE","DIG","DIM","DIN","DIP","DOE"
5130 DATA "DOG","DON","DOT","DRY","DUB","DUE","DUG"
5140 DATA "DYE","DUO","EAR","EAT","EGO","ELK","ELM"
5150 DATA "END","ELF","ERA","FAD","FAG","FAN","FAR"
5160 DATA "FAT","FED","FEW","FIG","FIN","FIR","FIT"
```

```
5170 DATA "FIX","FLY","FOE","FOG","FOR","FOX","FRY"
5180 DATA "FUN","FUR","GAP","GAS","GAY","GEM","GET"
5190 DATA "GIN","GNU","GOB","GOD","GOT","GUM","GUN"
5200 DATA "GUT","GUY","GYP","HAD","HAG","HAM","HAS"
5210 DATA "HAT","HAY","HEN","HEX","HID","HIM","HIP"
5220 DATA "HIS","HIT","HER","HUE","HUG","HUM","HOP"
5230 DATA "HOT","HOW","HUB","HEM","HOE","HOG","HUT"
5240 DATA "ICE","ICY","ILK","INK","IMP","ION","IRE"
5250 DATA "IRK","ITS","IVY","JAB","JAR","JAW","JAY"
5260 DATA "JOB","JOG","JOT","JOY","JUG","JAG","JAM"
5270 DATA "JET","JIB","JIG","JUT","KEG","KEY","KID"
5280 DATA "KIN","KIT","LAB","LAD","LAG","LAP","LAW"
5290 DATA "LAY","LAX","LED","LEG","LET","LID","LIE"
5300 DATA "LIP","LIT","LOB","LOP","LOT","LOW","LOG"
5310 DATA "LYE","MAD","MAN","MAP","MAR","MAT","MAY"
5320 DATA "MEN","MET","MID","MOB","MOP","MOW","MUD"
5330 DATA "MIX","MUG","NAB","NAG","NAP","NAY","NET"
5340 DATA "NEW","NIL","NIP","NOD","NOT","NOR","NOW"
5350 DATA "NUT","OAF","OAK","OAR","OAT","ODE","OIL"
5360 DATA "OLD","ONE","OPT","ORE","OUR","OUT","OVH"
5370 DATA "OWE","OWL","OWN","PAD","PAL","PAN","PAR"
5380 DATA "PAT","PAW","PAY","PEA","PEG","PEN","PET"
5390 DATA "PEW","PIE","PIG","PIT","PLY","POD","POT"
5400 DATA "POX","PER","PIN","PRO","PRY","PUB","PUN"
5410 DATA "PUS","PUT","RAG","RAM","RAN","RAP","RAT"
5420 DATA "RAW","RAY","RED","RIB","RID","REV","RIG"
5430 DATA "RIM","RIP","ROB","ROD","RuE","ROT","ROW"
5440 DHTA "RUB","RUE","RUG","RUM","RUN","RUT","RYE"
5450 DATA "SAD","SAG","SAP","SAT","SHW","SAY","SET"
5460 DATA "SEW","SEX","SHY","SEA","SIN","SHE","SIP"
5470 DATA "SIR","SIT","SIX","SKI","SKY","SLY","SOB"
5480 DATA "SOD","SON","SOW","SOY","SPA","SPY","STY"
5490 DATA "SUE","SUM","SUN","TAB","TAD","TAG","TAN"
5500 DATA "TAP","TAX","TAR","TEA","TEN","THE","THY"
5510 DATA "TIC","TIE","TIN","TIP","TOE","TON","TOP"
5520 DATA "TOW","TOY","TRY","TUB","TUG","TWO","URN"
5530 DATA "USE","UPS","VAN","VAT","VEX","VIA","VIE"
5540 DATA "VIM","VOW","YAK","YAM","YEN","YES","YET"
5550 DATA "YOU","WAD","WAG","WAN","WAR","WAS","WAX"
5560 DATA "WAY","WEB","WED","WET","WHO","WHY","WIG"
5570 DATA "WIN","WIT","WOE","WON","WRY","ZAP","ZIP"
9999 DATA "XXX"
```

EASY CHANGES

1. It's common for most players to request a summary before each guess. If you'd like to have a summary given automatically, change line 500 to:

   ```
   500 GOSUB 1000: PRINT:PRINT: PRINT "Your guess (";
   ```

2. The maximum number of guesses allowed, MG, can be changed in line 160. The current value of 25 is really somewhat larger than necessary – games rarely exceed 15 guesses. To set MG to 15, use:

   ```
   160 MG = 15:MW = 500
   ```

3. Modifying the data list of legal words is easy. The criteria used for the listed words were:

   No abbreviations
   No interjections ("ugh," "hey")
   No specialized words ("ohm," "yaw," etc.)
   No proper nouns

   Feel free to change the rules for your own word lists.

   In line 160, MW must be set to a value greater than the number of words in the data list. The list is contained in lines 5000-9999. Don't change line 9999, since it's the "stop flag."

MAIN ROUTINES

150-170	Dimensions arrays. Clears string space.
200-390	Initializes new game.
500-590	Human guesses the Sorcerer's word.
600-680	Sorcerer guesses.
800-870	Evaluates human's possible secret words. Moves them to the front of the WL$ array.
900-960	Processes inconsistency in given information.
1000-1120	Displays the current summary table.
1200-1240	Inquires about another game.
1500-1540	Compares a guess with key word.
1800-1830	Checks if input word is legal.
2200-2230	Shuffles WL$ array randomly.
2400-2410	Swaps elements P and Q in the WL$ array.
2600-2610	Breaks word Q$ into separate letters.
3000-3020	Fills WL$ array from data.
3200-3210	Post-mortem after Sorcerer wins.

3400-3490	Post-mortem after human wins.
3500-3550	List legal words.
3600-3620	Error routine – too many guesses.
5000-9999	Data.

MAIN VARIABLES

NW	Total number of data words.
MW	Size of WL$ array.
MG	Maximum number of guesses allowed.
WL$	String array holding data words.
HG$,CG$	String arrays of human's, Sorcerer's guesses.
HH,CH	Array of human's, Sorcerer's hits corresponding to HG$, CG$.
HNG,CNG	Current number of human's, Sorcerer's guesses.
CSW$	Sorcerer's secret word.
M1$,M2$, M3$	First, second, and third letters of a word.
W$,P$,Q$	String temporaries and work variables.
HPSW	Current number of human's possible secret words.
F	Flag for input word legality.
H	Number of hits in last guess.
W,K,J,P,Q	Temporaries; array and loop indices.
R$	User input.

SUGGESTED PROJECTS

1. Additional messages during the course of the game can personify the program even more. After the Sorcerer finds out how its last guess did, you might try an occasional message like one of these:

   JUST AS I THOUGHT . . .
   HMM, I DIDN'T EXPECT THAT . . .
   JUST WHAT I WAS HOPING TO HEAR . . .

   The value of HPSW is the number of words to which the computer has narrowed down the human's secret word. You might check its value regularly and, when it gets low, come out with something like

   BE CAREFUL, I'M CLOSING IN ON YOU.

2. Incorporate a feature to allow the loser to continue guessing at the other's word. The summary display routine will already work fine even if HNG and CNG are very different from each other. It will display a value of "9" for the number of hits corresponding to the correct guess of a secret word.
3. Try to speed up some of the program execution. It is slow during the initial array shuffling and after finding how its first few guesses did. There are several reasons for this. Try digging into the internal workings of the program with an eye toward lowering the execution time. Happy hunting!

WALLS

PURPOSE

This program allows you and a friend (or enemy) to play the game of WALLS, an arcade-like game that's one of our favorites. A combination of physical skills (reflex speed, hand-to-eye coordination, etc.) and strategic skills are needed to beat your opponent. Each game generally takes only a minute or two, so you'll want to play a match of several games to determine the better player.

HOW TO USE IT

The object of the game is to keep moving longer than your opponent without bumping into an obstacle. When the program starts, it asks in turn for the name of the player on the left and on the right. Then it displays the playing field, shows the starting point for each player, and tells you to press any key to start.

After a key is pressed, each player begins moving independently in one of four random directions—up, down, left, or right. As each player moves, he or she builds a "wall" inside the playing field. The computer determines the speed of the move; the player can only control his own direction. The player on the left can change direction to up, down, left, or right by pressing **W**, **Z**, **A**, or **D**, respectively. The player on the right does the same by using the keys **8**, **2**, **4**, and **6**. Find these keys on your

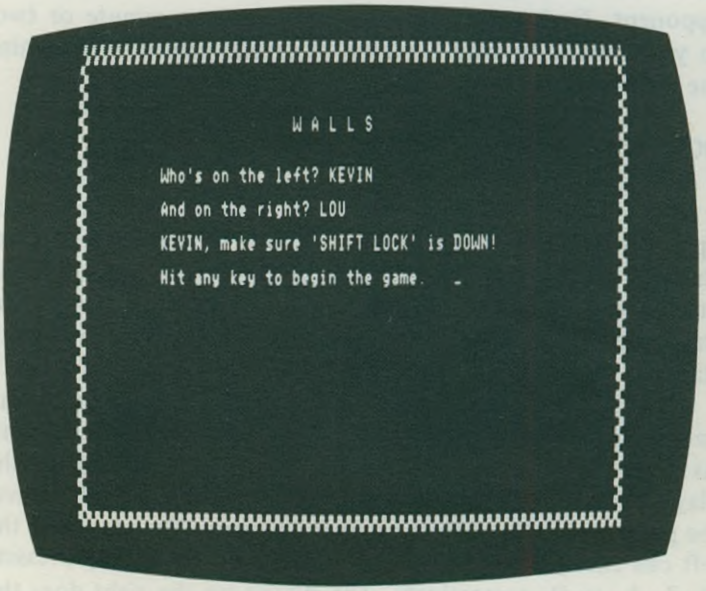
Sorcerer's keyboard and you will see the logic behind these choices. Note that **RETURN** isn't needed.

The first time either player bumps into the wall surrounding the playing field or the obstacle wall built by either player, he loses. When this happens, the program indicates the point of impact for a few seconds and displays the name of the winner. Then the game starts over. Ties are treated similarly.

The strategic considerations for this game are interesting. Should you attack your opponent, trying to build a wall around him into which he must crash? Or should you stay away from him and try to make efficient moves in an open area until your opponent runs out of room on his own? Try both approaches and see which yields the most success.

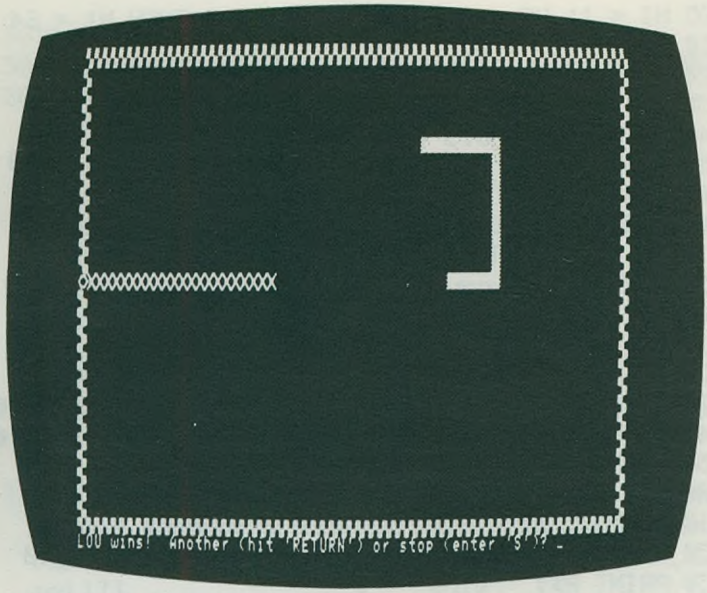
When pressing a key to change direction, be sure to press it quickly and release it. *Do not* hold it down—you might inhibit the computer from recognizing a move your opponent is trying to make. Once in a while, only one key will be recognized when two are hit at once.

SAMPLE RUN



```
                    W A L L S

      Who's on the left? KEVIN

      And on the right? LOU

      KEVIN, make sure 'SHIFT LOCK' is DOWN!

      Hit any key to begin the game.   _
```

The computer draws the playing field, then asks for the names of botn players. The game will begin when any key is pressed—but first, make sure that the **SHIFT LOCK** key is down.

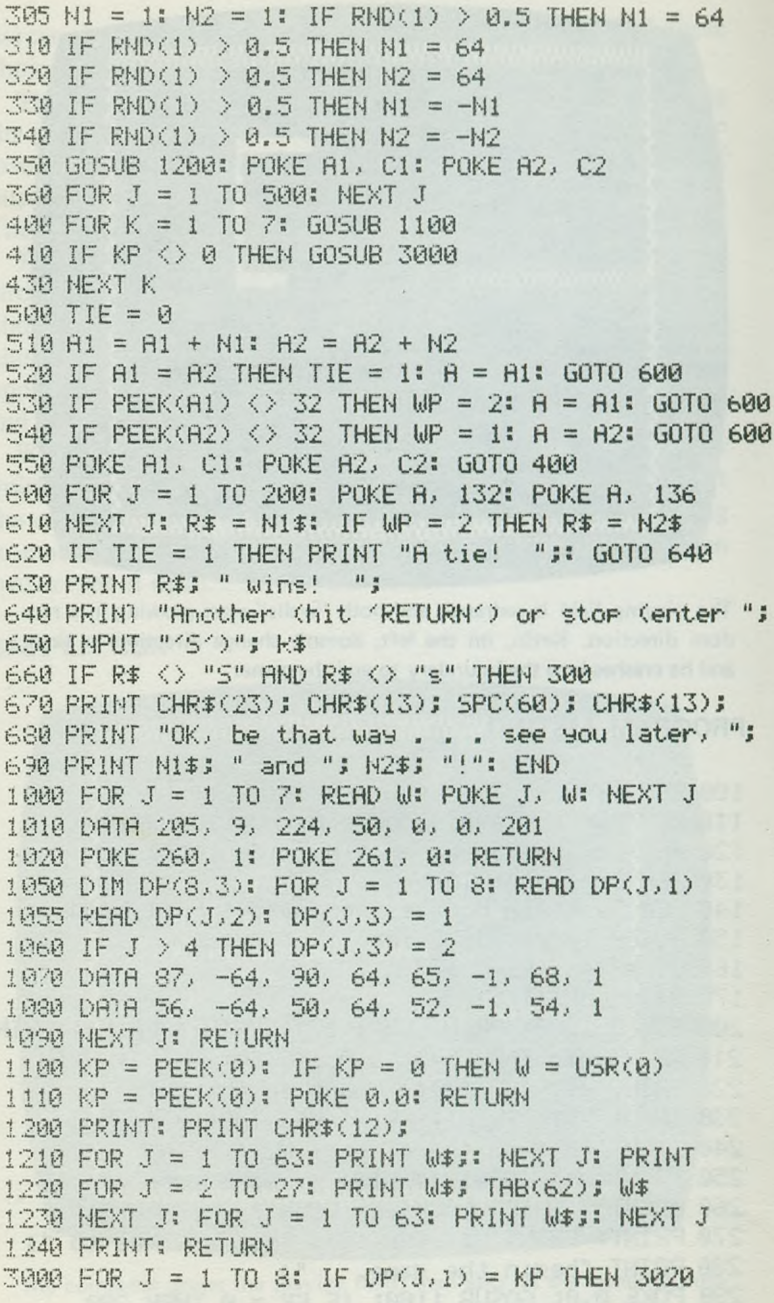


LOU wins! Another (hit 'RETURN') or stop (enter 'S')? _

The playing field is redrawn, and both "walls" start growing in a random direction. Kevin, on the left, doesn't change direction in time, and he crashes into the boundary to end the game.

PROGRAM LISTING

```
100 REM <WALLS>
110 REM Barrier avoidance/construction game
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 GOSUB 1000: GOSUB 1050
160 C1 = 142: C2 = 177
170 W$ = CHR$(165)
200 GOSUB 1200: PRINT CHR$(17): PRINT: PRINT: PRINT
210 PRINT W$; TAB(25); "W A L L S": PRINT: PRINT
220 PRINT W$; TAB(10); "Who's on the left";
230 INPUT N1$: PRINT: PRINT W$; TAB(10);
240 INPUT "And on the right"; N2$: PRINT
250 PRINT W$; TAB(10); N1$; ", make sure ";
260 PRINT "'SHIFT LOCK' is DOWN!"
270 PRINT: PRINT W$; TAB(10); "Hit any key to ";
280 PRINT "begin the game.  ";
290 POKE 0,0: GOSUB 1100: IF KP = 0 THEN 290
300 A1 = -3114: A2 = A1 + 20
```



```
305 N1 = 1: N2 = 1: IF RND(1) > 0.5 THEN N1 = 64
310 IF RND(1) > 0.5 THEN N1 = 64
320 IF RND(1) > 0.5 THEN N2 = 64
330 IF RND(1) > 0.5 THEN N1 = -N1
340 IF RND(1) > 0.5 THEN N2 = -N2
350 GOSUB 1200: POKE A1, C1: POKE A2, C2
360 FOR J = 1 TO 500: NEXT J
400 FOR K = 1 TO 7: GOSUB 1100
410 IF KP <> 0 THEN GOSUB 3000
430 NEXT K
500 TIE = 0
510 A1 = A1 + N1: A2 = A2 + N2
520 IF A1 = A2 THEN TIE = 1: A = A1: GOTO 600
530 IF PEEK(A1) <> 32 THEN WP = 2: A = A1: GOTO 600
540 IF PEEK(A2) <> 32 THEN WP = 1: A = A2: GOTO 600
550 POKE A1, C1: POKE A2, C2: GOTO 400
600 FOR J = 1 TO 200: POKE A, 132: POKE A, 136
610 NEXT J: R$ = N1$: IF WP = 2 THEN R$ = N2$
620 IF TIE = 1 THEN PRINT "A tie!  ";: GOTO 640
630 PRINT R$; " wins!  ";
640 PRINT "Another (hit 'RETURN') or stop (enter ";
650 INPUT "'S')"; R$
660 IF R$ <> "S" AND R$ <> "s" THEN 300
670 PRINT CHR$(23); CHR$(13); SPC(60); CHR$(13);
680 PRINT "OK, be that way . . . see you later, ";
690 PRINT N1$; " and "; N2$; "!": END
1000 FOR J = 1 TO 7: READ W: POKE J, W: NEXT J
1010 DATA 205, 9, 224, 50, 0, 0, 201
1020 POKE 260, 1: POKE 261, 0: RETURN
1050 DIM DP(8,3): FOR J = 1 TO 8: READ DP(J,1)
1055 READ DP(J,2): DP(J,3) = 1
1060 IF J > 4 THEN DP(J,3) = 2
1070 DATA 87, -64, 90, 64, 65, -1, 68, 1
1080 DATA 56, -64, 50, 64, 52, -1, 54, 1
1090 NEXT J: RETURN
1100 KP = PEEK(0): IF KP = 0 THEN W = USR(0)
1110 KP = PEEK(0): POKE 0,0: RETURN
1200 PRINT: PRINT CHR$(12);
1210 FOR J = 1 TO 63: PRINT W$;: NEXT J: PRINT
1220 FOR J = 2 TO 27: PRINT W$; TAB(62); W$
1230 NEXT J: FOR J = 1 TO 63: PRINT W$;: NEXT J
1240 PRINT: RETURN
3000 FOR J = 1 TO 8: IF DP(J,1) = KP THEN 3020
```

```
3010 NEXT J: RETURN
3020 P = DP(J,3): IF P = 1 THEN N1 = DP(J,2):RETURN
3030 N2 = DP(J,2): RETURN
```

EASY CHANGES

1. To speed up the game, change the 7 in line 400 to a 4 or 5. To slow the game, change it to 12 or 15.
2. To make both players always start moving towards the top, insert:

   305 N1 = −64: N2 = −64: GOTO 350

   To make both players start moving towards each other, use:

   305 N1 = 1: N2 = −1: GOTO 350
3. The graphics display characters for each player can be easily changed. Simply alter the value for C1 (left side) and /or C2 (right side) in line 160. Good values to try include 132, 136, and 173.
4. To change the length of time that the "crash" dot flashes at the end of the game, modify line 600. Change the constant 200 to larger values to lengthen the time, or to 100 or 150 to shorten it.

MAIN ROUTINES

150-170 Initializes graphics. Sets up key-detection subroutine and direction/player array DP.

200-290 Draws playing field. Gets players' names.

300-360 Sets up initial positions and directional increments. Redraws playing field, then pauses before starting game.

400-430 Searches for a pressed key. If found, key value is checked against DP array directional increments.

500-550 Checks for a win or tie. If none, display updated and search for pressed key repeats.

600-690 Prints ending messages. Gets user's request to stop or play again.

1000-1020 Subroutine to set up machine language USR input routine.

1050-1090 Subroutine to set up direction/player array DP.
1100-1110 Subroutine to check for pressed key, with ASCII value of key stored at address zero by the USR routine.
1200-1240 Subroutine to draw playing field.
3000-3030 Subroutine to search DP array for values corresponding to ASCII keyboard input.

MAIN VARIABLES

C1,C2 ASCII values for left and right players' wall characters.
W$ Graphics value for perimeter walls.
N1$,N2$ Players' names.
KP ASCII value of key pressed.
N1,N2 Current increment for updating end-of-wall POKE address (up = −64, down = 64, left = −1, right = 1).
A1,A2 POKE address for ends of walls.
TIE Tied game flag (set to 1 if tie occurs).
R$ Work string variable, and user input.
J,K,W Loop counters, work variables.
WP Number of winning player.
A Address of collision point.
P Player number.
DP Array to hold eight ASCII values of valid direction key inputs–DP(n,1)–corresponding increments for POKE values–DP(n,2)–and player number–DP(n,3).

SUGGESTED PROJECTS

1. Change the size of the playing field. The 63 in lines 1210 and 1230 is the width, and the 27 in line 1220 is the height. Note that line 300 has the starting addresses of the two players. You may want to change these if you make the field smaller.
2. Keep score over a seven-game (or so) match. Display the current score after each game. Don't forget to allow for ties.
3. Modify the program to let each player press only two keys–one to turn left from the current direction of travel, and one to turn right.

4. Instead of a game between two people, make it a game of a person against the computer. Develop a computer strategy to keep finding open areas to move to and/or cut off open areas from the human opponent.
5. The key-detection mechanism, set up in lines 1000-1020 and run in lines 1100-1110, replaces the GET command found on some machines. A USR command calls the machine language subroutine located at addresses 1-7 and 260-261. This routine checks an internal CPU register for key values, and transfers that value to address zero. A PEEK can then get that value, or see that no new value has been found.

   As written, the Sorcerer monitor's general-purpose RECEVE checks whatever device is assigned as the console input—the keyboard, unless the SET I =[device] command is used. The search can be speeded using the keyboard-only routine KEYBRD rather then RECEVE. Simply change the second data item in line 1010 from 9 to 24 to use KEYBRD.

RACER

PURPOSE

Imagine yourself at the wheel of a high-speed race car winding your way along a treacherous course. The road curves unpredictably. To stay on course, you must steer accurately or risk collision. How far can you go in one day? How many days will it take you to race cross-country? Thrills galore without leaving your living room.

The difficulty of the game is completely under your control. By adjusting the road width and visibility conditions, RACER can be made as easy or as challenging as you wish.

HOW TO USE IT

The program begins with a request for a random value. It then asks you for two inputs: road width and visibility. The road width can be set anywhere between 1 and 9. The degree of difficulty changes appreciably with different widths. A very narrow setting will be quite difficult and a wide one relatively easy. Visibility can be set to any of five settings, ranging from terrible to good. When visibility is good, the car appears high on the screen. This allows a good view of the twisting road ahead. When visibility is poor, the car appears low on the screen, allowing only a brief look at the upcoming road.

Having set road width and visibility, the race is ready to start. The car appears on the road at the starting line. A 6-step starting light counts down the start. When the bottom lights go on, the race begins. The road moves continually up the screen.

Its twists and turns are controlled randomly. You must steer the car accurately to keep it on track.

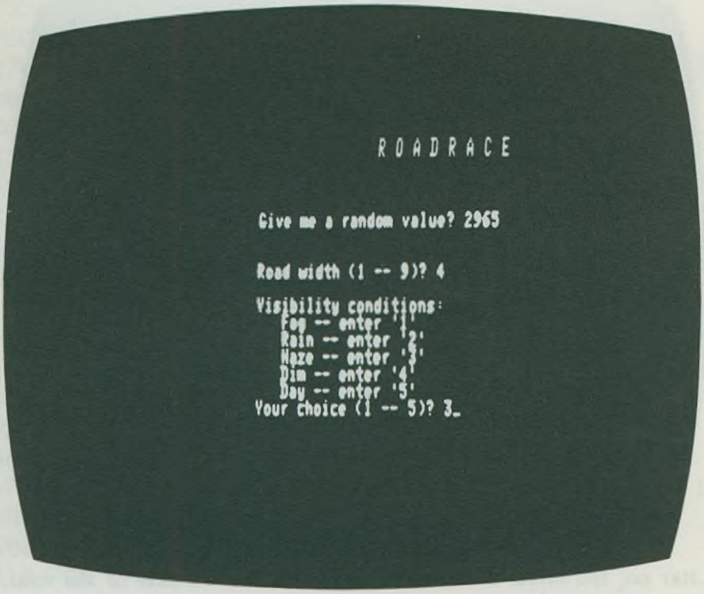
The car is controlled with the use of two keys on the numeric keypad. Pressing the **4** will cause the car to move to the left while pressing the **6** will cause a move to the right. Doing neither will cause the car to continue straight down. Use short "stabs" at the keys.

The race proceeds until the car goes "off the road". Each such collision is considered to terminate one day of the race. After each day, you are shown the number of miles achieved that day along with the cumulative miles achieved for consecutive days of the race.

After each collision, you can proceed by entering either **C**, **R**, or **S**. Selecting **C** will continue the race for another day with the same road conditions. Cumulative totals will be retained. **R** will restart the race. This allows changing the road conditions and initializing back to day one. **S** simply quits the race and returns the Sorcerer back to BASIC.

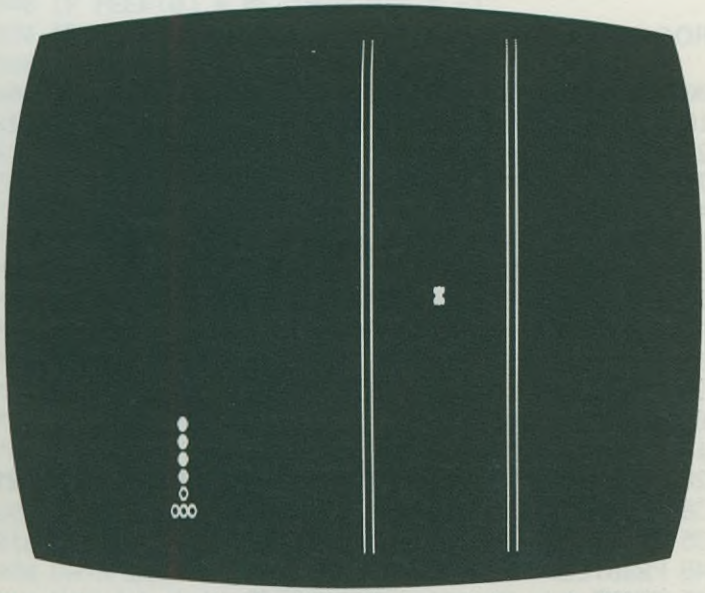
There are several different ways to challenge yourself with the program. You can try to see how far you get in a given number of days. You might see how many days it takes you to go a given number of miles—say 300 miles for a cross-country trip. As you become proficient at one set of road conditions, make the road narrower and/or the visibility poorer. This will increase the challenge. Different road conditions can also be used as a handicapping aid for two unequally matched opponents.

SAMPLE RUN

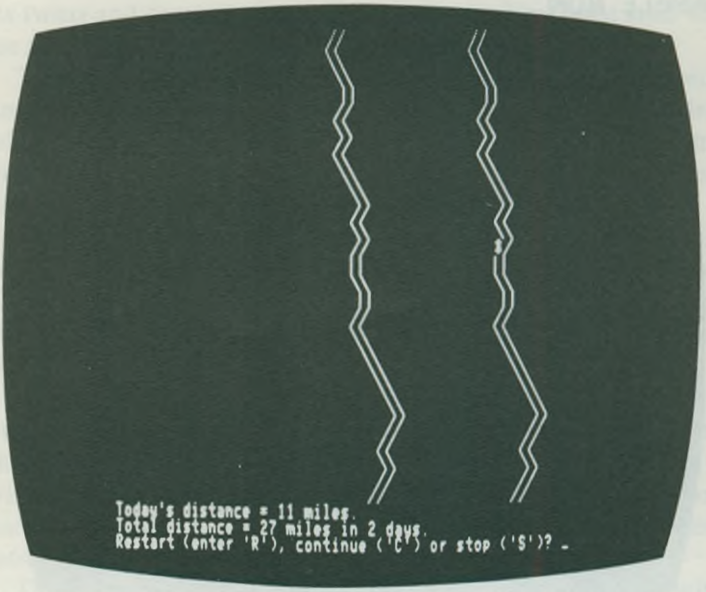


```
                    R O A D R A C E


      Give me a random value? 2965

      Road width (1 -- 9)? 4

      Visibility conditions:
         Fog -- enter '1'
         Rain -- enter '2'
         Haze -- enter '3'
         Dim -- enter '4'
         Day -- enter '5'
      Your choice (1 -- 5)? 3_
```

The short input phase requires a random value, and visibility and road width specifications.



The car is on the starting line. When the lowest lights on the "Christmas tree" turn on, the race will begin.



```
Today's distance = 11 miles.
Total distance = 27 miles in 2 days.
Restart (enter 'R'), continue ('C') or stop ('S')? _
```

Later on, the driver crashes into the barrier at the side of the road. Total distance for this and prior "days" are displayed, along with options for continuing the game.

PROGRAM LISTING

```
100 REM <RACER>
110 REM Roadrace arcade game
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 LCUT = 0.4: RCUT = 1 - LCUT
160 ML = 52: MR = 54
170 CL$=CHR$(171):CR$=CHR$(172):CS$=CHR$(128)
180 GOSUB 900: PRINT CHR$(12), "R O A D R A C E"
185 PRINT: PRINT: PRINT: PRINT "Give me a random ";
190 INPUT "value"; J: PRINT: PRINT
200 TDIST = 0: NDAY = 0
210 INPUT "Road width (1 -- 9)"; WID
220 WID = INT(WID): IF WID < 1 OR WID > 9 THEN 210
230 PRINT: PRINT "Visibility conditions:"
235 PRINT "   Fog -- enter '1'"
240 PRINT "   Rain -- enter '2'"
245 PRINT "   Haze -- enter '3'"
```

```
250 PRINT "  Dim -- enter '4'"
255 PRINT "  Day -- enter '5'"
260 INPUT "Your choice (1 -- 5)"; VIS
270 VIS = INT(VIS): IF VIS < 1 OR VIS > 5 THEN 260
280 J = RND(-ABS(J))
300 NDAY = NDAY + 1: DDIST = 0: R$ = CS$
310 LPO = 32 - 2 * WID: RPO = LPO + 4 * WID
320 FOR J = 1 TO 30: GOSUB 800: NEXT J
330 CPO = -2080 - 320 * VIS: POKE CPO, 192
340 GOSUB 1000
400 K = RND(1): CND = 0: LR$ = R$: R$ = CS$
410 IF K > LCUT OR LPO < 3 THEN 440
420 IF LR$ = CR$ THEN LPO = LPO + 1
430 CND = -1: R$ = CL$: GOTO 470
440 IF K < RCUT OR RPO > 61 THEN 470
450 CND = 1: R$ = CR$
460 IF LR$ <> CR$ THEN LPO = LPO - 1
470 IF LR$ = CR$ AND R$ = CS$ THEN LPO = LPO + 1
480 LPO = LPO + CND: RPO = LPO + 4 * WID
500 DIR = 0: POKE 0,0
510 FOR J = 1 TO 7: K = USR(0)
520 IF PEEK(0) <> 0 THEN 540
530 NEXT J: GOTO 560
540 IF PEEK(0) = ML THEN DIR = -1
550 IF PEEK(0) = MR THEN DIR = 1
560 POKE CPO, 32: CPO = CPO + DIR
600 IF PEEK(CPO + 64) <> 32 THEN 700
610 GOSUB 800: DDIST = DDIST + 1: POKE CPO, 192
620 GOTO 400
700 GOSUB 800: POKE CPO, 42: TDIST = TDIST + DDIST
710 FOR J = 1 TO 500: NEXT J
720 PRINT "Today's distance ="; INT(DDIST / 10);
730 PRINT "miles.": PRINT "Total distance =";
740 PRINT INT(TDIST/10); "miles in"; NDAY; "days."
750 PRINT "Restart (enter 'R'), continue ";
760 INPUT "('C') or stop ('S')"; W$: PRINT
770 IF W$ = "S" OR W$ = "s" THEN END
775 IF W$ = "R" OR W$ = "r" THEN 200
780 IF W$ = "C" OR W$ = "c" THEN 300
790 PRINT "What?": GOTO 750
800 PRINT TAB(LPO); R$; R$; TAB(RPO); R$; R$
810 RETURN
900 DATA 205, 9, 224, 50, 0, 0, 201: RESTORE
910 FOR J = 1 TO 7: READ K: POKE J, K: NEXT J
```

```
920 DATA 1, 0, 153, 255, 153, 60, 60, 153, 255, 153
930 FOR J = 260 TO 261: READ K: POKE J, K: NEXT J
940 FOR J = -512 TO -505: READ K: POKE J, K: NEXT J
950 RETURN
1000 FOR J = -2621 TO -2301 STEP 64: POKE J, 136
1010 NEXT J: POKE -2302, 136: POKE -2300, 136
1020 FOR J = 1 TO 250: NEXT J
1030 FOR J = -2621 TO -2301 STEP 64
1040 FOR K = 1 TO 200: NEXT K: POKE J, 132: NEXT J
1050 POKE -2302, 132: POKE -2300, 132: RETURN
```

EASY CHANGES

1. The keys which cause the car to move left and right can be easily changed. You may wish to do this if you are left-handed or find that two widely separated keys would be more convenient. The changes are to be made in line 160. Left and right movements are controlled by the two ASCII values in variables ML and MR. If, for example, you wanted A to cause a left move and 3 to cause a right move, change line 160 to read

   160 ML=65:MR=51

2. The amount of windiness in the road can be adjusted by changing the value of LCUT in line 150. Maximum windiness is achieved with a value of 0.5 for LCUT. To get a straighter road, make LCUT smaller. A value of zero will produce a completely straight road. LCUT should lie between zero and 0.5; otherwise the road will drift to one side and linger there. To get a somewhat windier road, you might change line 150 to read

   150 LCUT=0.45:RCUT=1 − LCUT

MAIN ROUTINES

150-190	Variable initialization.
200-280	Gets road conditions from user.
300-340	Initializes the road.
400-480	Determines the next road condition.
500-620	Updates the car position.
700-790	Processes end of race day.
800-810	Draws next road segment (subroutine).

900-950	Set up USR routine, car character (subroutine).
1000-1050	Graphics to begin race (subroutine).

MAIN VARIABLES

WID	Road width.
VIS	Visibility.
DDIST	Distance driven on current day.
NDAY	Number of days of the race.
TDIST	Total distance driven for whole race.
ML,MR	ASCII values of characters to move car left, right.
LPO,RPO	Position of left, right side of road.
LCUT,RCUT	Random value cutoff to move road left, right.
W$	User replies.
CPO	Position of car.
CL$,CR$, CS$	Characters for road segments.
J,K	Loop indices and work variables.
R$,LR$	Current, previous road border characters.
DIR	Direction of requested movement.
CND	Direction of random road movement.

SUGGESTED PROJECTS

1. Write a routine to evaluate a player's performance after each collision. Display a message rating him anywhere from "expert" to "back seat driver." This should involve comparing his actual miles achieved against an expected (or average) number of miles for the given road width and visibility. For starters, you might use

   Expected miles = (WID ↑ 3)/10 + VIS

   This formula is crude, at best. The coding can be done between lines 450 and 460.
2. Incorporate provisions for two players racing one at a time. Keep cumulative totals separately. After each collision, display the current leader and how far he is ahead.
3. Add physical obstacles or other hazards onto the road in order to increase the challenge. This can be done with ran-

dom additions to the output of lines 800-810. The program will recognize a collision if the car moves into any nonblank square.

4. The key-detection USR routine is identical to the method explained in WALLS. Use additional keys as "accelerators" or "brakes".

WARI

PURPOSE

Wari is an old game with roots that are even older. Its origins go back thousands of years to a variety of other similar games, all classified as being members of the Mancala family. Among the countless variations are Awari, Oware, Pallanguli, and Kalah.

The program matches you against the computer. You are probably going to lose a few games before you win one – the computer plays a pretty good game. This may hurt your ego a little bit, since Wari is purely a skill game (like chess or checkers). There is no element of luck involved, as would be the case with backgammon, for example. When you lose, it's because you were outplayed.

HOW TO USE IT

When you start the program, it begins by asking if you want to go first. The board is made up of twelve squares in two rows of six. Your side is on the bottom, numbered one through six from left to right. The computer's side is on the top, numbered seven through twelve from right to left.

At the start of the game, each square has four "stones" in it. There is no way to differentiate between your stones and the computer's. They all look alike and will move from one side to the other during the course of play.

The first player "picks up" all the stones in one of the squares on his side of the board and drops them, one to a square, starting with the next numbered square. The stones continue to be dropped consecutively in each square, continuing over onto the opponent's side if necessary (after square number 12 comes square number 1 again).

If the last stone is dropped onto the opponent's side *and* leaves a total of either two or three stones in that square, these stones are captured by the player who moved, and are removed from the board. Also, if the next-to-last square in which a stone was dropped meets the same conditions (on the opponent's side and now with two or three stones), its stones are also captured. This continues backwards until the string of consecutive squares with two or three stones on the opponent's side is broken.

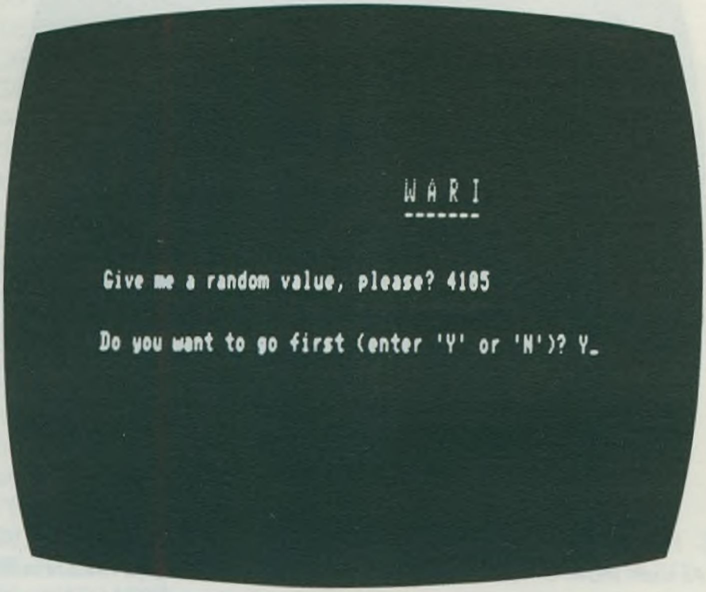
Regardless of whether any captures are made, play alternates back and forth between the two players. The object of the game is to be the first player to capture 24 or more stones. That's half of the 48 stones that are on the board at the beginning of the game.

There are a few special rules to cover some situations that can come up in the game. It is not legal to capture all the stones on the opponent's side of the board, since this would leave the opponent with no moves on his next turn. By the same token, when your opponent has no stones on his side (because he had to move his last one to your side on his turn), you have to make a move that gives him at least one stone to move on his next turn, if possible. If you cannot make such a move, the game is over and counted as a draw.

During the course of the game, it's possible for a square to accumulate twelve or more stones in it. Moving from such a square causes stones to be distributed all the way around the board. When this happens, the square from which the move was made is skipped over, and is always left empty.

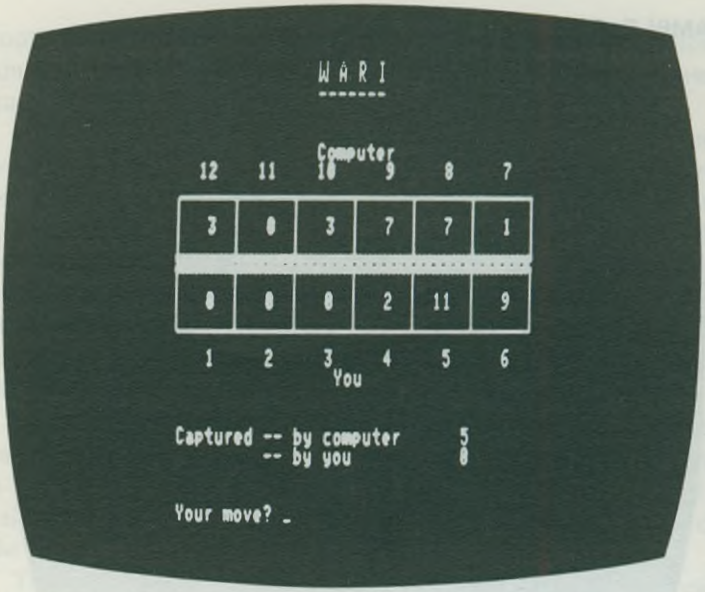
It takes the computer anywhere from five seconds to about thirty seconds to make a move, depending on the complexity of the board position. The word THINKING is displayed during this time, and a period is added to it as each possible move is evaluated in sequence (seven through twelve).

SAMPLE RUN



```
                              W A R I
                              -------

     Give me a random value, please? 4105

     Do you want to go first (enter 'Y' or 'N')? Y_
```

A short introductory phase asks for a random value. The user can select whether he or the Sorcerer will go first.



```
                    W A R I
                    -------

                    Computer
        12     11     10     9     8     7

        3      0      3      7     7     1

        0      0      0      2     11    9

        1      2      3      4     5     6
                     You

        Captured -- by computer      5
                 -- by you           0

        Your move? _
```

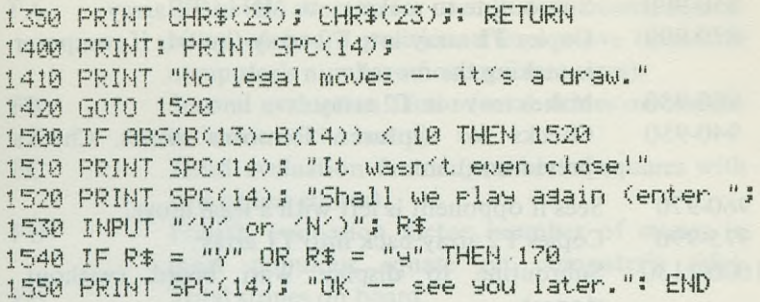
As each move is input, the board is changed accordingly. The Sorcerer then evaluates its possible moves, selecting the best available. Later in the same game, the Sorcerer has captured five men, while shutting out its opponent.

PROGRAM LISTING

```
100 REM <WARI>
110 REM Wari -- a board game
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 DIM B(14), T1(14), T2(14), T3(14)
160 DIM EOP(6), ECO(6)
170 NMOVE = 0: FOR J = 1 TO 12: B(J) = 4: NEXT J
180 PRINT CHR$(12),,"W A R I": PRINT,,"-------"
190 B(13) = 0: B(14) = 0: MN = 0: PRINT: PRINT
200 INPUT "Give me a random value, please"; W
210 F1 = RND(-ABS(W)): F2 = RND(1) / 14
220 F1 = F2 + 0.25: F2 = 0.25 - F1: PRINT: PRINT
230 PRINT "Do you want to go first (enter 'Y' or";
240 INPUT " 'N')"; R$: R$ = LEFT$(R$,1): PRINT
250 IF R$ = "Y" OR R$ = "y" THEN 270
260 IF R$ <> "N" AND R$ <> "n" THEN 230
270 GOSUB 1000: GOSUB 1200
```

```
280 IF R$ = "Y" OR R$ = "y" THEN 400
300 GOSUB 1300: PRINT SPC(20); "Thinking";
310 GOSUB 500: IF MF < 1 THEN 1400
320 PRINT " My move is square"; MF
330 FOR J = 1 TO 14: T1(J) = B(J): NEXT J
340 GOSUB 850
350 FOR J = 1 TO 14: B(J) = T1(J): NEXT J
360 GOSUB 1200: IF B(14) < 24 THEN 390
370 PRINT SPC(14); "I won"; B(14); "to"; B(13);"."
380 GOTO 1500
390 FOR J = 1 TO 3000: NEXT J
400 GOSUB 1300: PRINT SPC(20);
410 INPUT "Your move"; R$: MF = INT(VAL(R$))
420 IF MF > 0 AND MF < 7 THEN 450
430 PRINT SPC(14); "Sorry -- illegal move!"
440 FOR J = 1 TO 3000: NEXT J: GOTO 400
450 FOR J = 1 TO 14: T1(J) = B(J): NEXT J
460 GOSUB 850: IF MF < 0 THEN 430
470 FOR J = 1 TO 14: B(J) = T1(J): NEXT J
480 GOSUB 1200: NMOVE=NMOVE+1: IF B(13)<24 THEN 300
485 PRINT SPC(14); "You won"; B(13);"to";B(14);"."
490 GOTO 1500
500 FOR MN = 1 TO 6: MF = MN + 6
510 IF B(MF) = 0 THEN ECO(MN) = -50: GOTO 700
520 FOR J=1 TO 14: T1(J) = B(J): NEXT J: GOSUB 850
530 IF MF < 0 THEN ECO(MN) = -50: GOTO 700
540 IF T1(14) > 23 THEN MF = MN + 6: RETURN
550 FOR J=1 TO 14:T3(J)=T1(J): NEXT J: FOR K=1 TO 6
560 IF T1(K) = 0 THEN EOP(K) = 50: GOTO 670
570 FOR J=1 TO 14:T1(J)=T3(J):NEXT J:MF=K:GOSUB 850
580 IF MF < 0 THEN T2(K) = 50: GUTO 670
590 FA = 0: FB = 0: FC = 0: FD = 0: FOR J = 7 TO 12
600 FB = FB + T1(J): IF T1(J) > 0 THEN FA = FA + 1
610 IF T1(J) < 3 THEN FC = FC + 1
620 IF T1(J) > FD THEN FD = T1(J)
630 NEXT J: FE=FB: FOR J=1 TO 6: FE=FE+T1(J):NEXT J
640 FA=FA/6: FD=1-FD/FB: FC=1-FC/6: FB=FB/FE
650 EOP(K) = F1 * (FA+FB) + F2 * (FC+FD)
660 EUP(K)=EOP(K) + T1(14) - T1(13) + B(13) - B(14)
670 NEXT K: ECO(MN) = 50: FOR J = 1 TO 6
680 IF EOP(J) < ECO(MN) THEN ECO(MN) = EOP(J)
690 NEXT J
700 PRINT ".";: NEXT MN: MF=0: MN=-50: FOR J=1 TO 6
710 IF ECO(J) > MN THEN MN = ECO(J): MF = J + 6
720 NEXT J: RETURN
```

```
850 IF T1(MF) = 0 THEN MF = -1: RETURN
860 R$ = "H": IF MF > 6 THEN R$ = "C": GOTO 880
870 FOR J = 1 TO 14: T2(J) = T1(J):NEXT J:GOTO 900
880 FOR J = 1 TO 6: T2(J) = T1(J+6):T2(J+6) = T1(J)
890 NEXT J: T2(13)=T1(14): T2(14)=T1(13): MF = MF-6
900 C = MF: N = T2(C): FOR J = 1 TO N: C = C + 1
910 IF C = 13 THEN C = 1
920 IF C = MF THEN C = C + 1: GOTO 910
930 T2(C) = T2(C) + 1: NEXT J: T2(MF) = 0: LS = C
940 IF LS < 7 OR T2(LS)-> 3 OR T2(LS) < 2 THEN 960
950 T2(13)=T2(13)+T2(LS):T2(LS)=0:LS=LS-1:GOTO 940
960 OS = 0: FOR J = 7 TO 12: OS = OS + T2(J): NEXT
970 IF OS = 0 THEN MF = -2: RETURN
975 IF R$ <> "H" THEN 985
980 FOR J = 1 TO 14: T1(J) = T2(J): NEXT J: RETURN
985 FOR J=1 TO 6: T1(J) = T2(J+6): T1(J+6) = T2(J)
990 NEXT J:T1(14) = T2(13): T1(13) = T2(14):RETURN
1000 PRINT CHR$(12); TAB(28); "W A R I"
1010 PRINT TAB(28); "-------": PRINT: PRINT
1020 PRINT TAB(28); "Computer": PRINT TAB(16);
1030 PRINT "12    11    10     9     8     7"
1040 PRINT TAB(14);: FOR J = 15 TO 50
1050 PRINT CHR$(176);: NEXT J: PRINT: FOR J=1 TO 3
1060 GOSUB 1120: NEXT J: PRINT TAB(13); CHR$(135);
1070 FOR J = 15 TO 50: PRINT CHR$(192);: NEXT J
1080 PRINT: FOR J = 1 TO 3: GOSUB 1120: NEXT J
1090 PRINT TAB(14);: FOR J = 15 TO 50
1100 PRINT CHR$(137);: NEXT J: PRINT:PRINT TAB(17);
1110 PRINT "1     2     3     4     5     6"
1115 PRINT TAB(30); "You": RETURN
1120 PRINT TAB(13);: FOR K = 1 TO 7
1130 PRINT CHR$(135); SPC(5);: NEXT K: PRINT:RETURN
1200 FOR J = 1 TO 6: HD = -3190 + 6*J
1210 GOSUB 1240: NEXT J
1220 FOR J = 7 TO 12: AD = -3368 - 6*J
1230 GOSUB 1240: NEXT J: RETURN
1240 W = INT(B(J) / 10)
1250 IF W = 0 THEN POKE AD, 32: GOTO 1270
1260 POKE AD, W+48
1270 W = B(J) - 10*W: POKE AD+1, W+48: RETURN
1300 PRINT CHR$(17);: FOR J = 1 TO 19: PRINT:NEXT J
1310 PRINT TAB(14); "Captured -- by computer";
1320 PRINT TAB(42); B(14): PRINT TAB(23);
1330 PRINT "-- by you"; TAB(42); B(13): PRINT
1340 FOR J = 1 TO 3: PRINT SPC(62): NEXT J
```



```
1350 PRINT CHR$(23); CHR$(23);: RETURN
1400 PRINT: PRINT SPC(14);
1410 PRINT "No legal moves -- it's a draw."
1420 GOTO 1520
1500 IF ABS(B(13)-B(14)) < 10 THEN 1520
1510 PRINT SPC(14); "It wasn't even close!"
1520 PRINT SPC(14); "Shall we play again (enter ";
1530 INPUT "'Y' or 'N')"; R$
1540 IF R$ = "Y" OR R$ = "y" THEN 170
1550 PRINT SPC(14); "OK -- see you later.": END
```

EASY CHANGES

1. Change the length of time the "illegal" message is displayed after an improper move is attempted by changing the number 3000 in line 440. Double it to double the length of time, etc.
2. If you are curious about what the computer thinks are the relative merits of each of its possible moves, you can make this change to find out. Change line 700 so it looks like this:

   700 PRINT ECO(MN);:NEXT
   MN:MF=0:MN=−50:FOR J=1 TO 6

   This will cause the program to display its evaluation number for each of its moves in turn (starting with square seven). It will select the largest number of the six. A negative value means that it will lose stones if that move is made, assuming that you make the best reply you can. A value of negative 50 indicates an illegal move. A positive value greater than one means that a capture can be made.

MAIN ROUTINES

150-190	Initializes variables.
200-280	Asks who goes first. Evaluates answer. Displays board.
300-390	Determines computer's move. Displays new board position. Determines if computer's move resulted in a win. Displays a message if so.
400-490	Gets operator's move. Checks for legality. Displays new board position. Determines if operator's move resulted in a win.
500-720	Subroutine to determine computer's move.

850-990 Subroutine to make move MN in T1 array.
870-890 Copies T1 array into T2 array (inverts if computer is making the move).
900-930 Makes move in T2 array.
940-950 Checks for captures. Removes stones. Checks previous square.
960-970 Sees if opponent is left with a legal move.
975-990 Copies T2 array back into T1 array.
1000-1130 Subroutine to display Wari board (without stones).
1200-1270 Subroutine to display stones on board.
1300-1350 Subroutine to move cursor.
1400-1420 Displays messages when computer has no legal move.
1500-1550 Gets continuation option.

MAIN VARIABLES

J,K Subscript variables.
T1,T2,T3 Arrays with temporary copies of the Wari board.
EOP Array with evaluation values of operator's six possible replies to computer's move being considered.
ECO Array with evaluation values of computer's six possible moves.
B Array containing Wari board. Thirteenth element has stones captured by operator. Fourteenth has computer's.
F1,F2 Weighting factors for evaluation function.
NMOVE Move number.
R$ Operator's reply. Also used as switch to indicate whose move it is.
MF Move being made (1-6 for operator, 7-12 for computer). Set negative if illegal.
C Subscript used in dropping stones around board.
LS Last square in which a stone was dropped.
OS Stones on opponent's side of the board after a move.
MN Subscript used to indicate which of the six possible computer moves is currently being evaluated.

FA First evaluation factor used in determining favorability of board position after a move (indicates computer's number of occupied squares).

FB Second evaluation factor (total stones on computer's side of the board).

FC Third evaluation factor (number of squares with two or less stones).

FD Fourth evaluation factor (number of stones in most populous square on computer's side).

FE Total stones on board.

W,N Work variables.

SUGGESTED PROJECTS

1. Modify the program to declare the game a draw if neither player has made a capture in the past thirty moves. Line 480 adds one to the counter of the number of moves made. To make the change, keep track of the move number of the last capture, and compare the difference between it and the current move number with 30.
2. Modify the evaluation function used by the computer strategy to see if you can improve the quality of its play. Lines 590 through 660 examine the position of the board after the move that is being considered. Experiment with the factors and/or the weighting values, or add a new factor of your own.
3. Change the program so it can allow two people to play against each other, instead of just a person against the computer.

Section 4
Graphics Display Programs

INTRODUCTION TO GRAPHICS DISPLAY PROGRAMS

The Sorcerer is an amazing machine. It has special graphics capabilities not found on other similar computers. Programs in the other sections of this book take advantage of these graphics to "spice up" output. Here we explore the use of the Sorcerer's graphic capabilities for sheer fun, amusement, and diversion.

Ever look through a kaleidoscope and enjoy the symmetric changing patterns produced? KLEID will create such effects with full eight-point symmetry.

Two other programs produce ever-changing patterns but with much different effects. SPARK will fascinate you with a changing shimmering collage. SQUAR uses geometric shapes to obtain its pleasing displays.

ACROB demonstrates a totally different aspect of the Sorcerer's graphic abilities. This program will keep you entertained with an example of computer animation.

KALEIDO

PURPOSE

If you have ever played with a kaleidoscope, you were probably fascinated by the endless symmetrical patterns you saw displayed. This program creates a series of kaleidoscope-like designs, with each one overlaying the previous one. The designs are symmetrical about eight axes—can you find them?

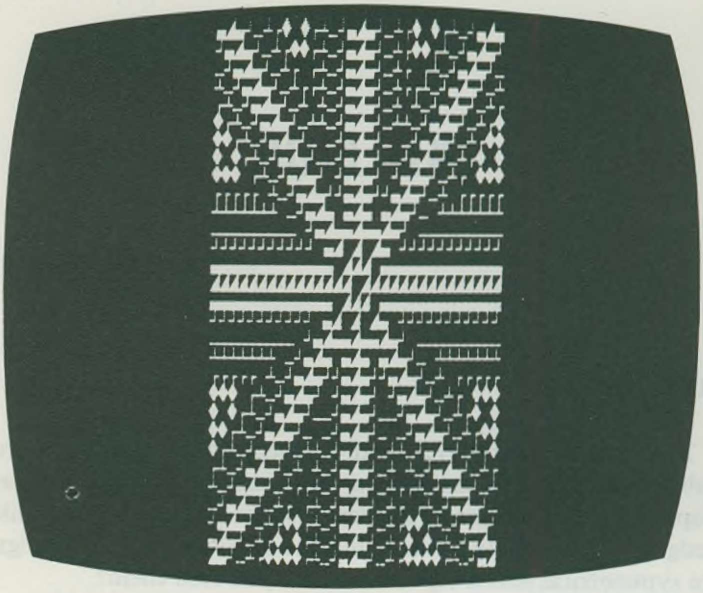
HOW TO USE IT

There is not much to say about how to use this one. Just type **RUN**, then sit back and watch. Turning down the lights and playing a little music is a good way to add to the effect.

By the way, it is a little misleading to say that the designs you see are symmetrical. It is more accurate to say that the positions occupied by the individual graphics characters are located symmetrically. The overall design is usually not completely symmetrical, since the individual graphics characters are not themselves symmetrical. The characters on the lower half of the design would have to be the upside-down equivalent of the corresponding characters on the upper half for that to be true.

Have a few friends bring their Sorcerers over (all your friends *do* have Sorcerers, don't they?), and get them all going with KLEID at once. Let us know if you think you have set a new world's record. Please note that we will not be responsible for any hypnotic trances induced this way. To stop—if you aren't hypnotized—hit **CTRL C**.

SAMPLE RUN



One of the endless kaleidoscopic patterns created by KLEID.

PROGRAM LISTING

```
100 REM <KLEID>
110 REM Kaleidoscopic graphics
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 CX = 32: CY = 15: R = RND(-1): D = -1: DIM A(6)
160 T = 128: NC = 64
170 PRINT CHR$(12);
200 FOR J=0 TO 6: A(J) = T + INT(NC*RND(1)): NEXT J
300 D = 1
310 K = 1: L = 15: IF D < 0 THEN K = 15: L = 1
400 FOR J = K TO L STEP D
410 X = CX + J: Y = CY: GOSUB 1000
420 X = CX - J: GOSUB 1000
430 X = CX: Y = CY + J: GOSUB 1000
440 Y = CY - J: GOSUB 1000
450 X = CX + J: Y = CY + J: GOSUB 1000
460 X = CX - J: Y = CY - J: GOSUB 1000
470 Y = CY + J: GOSUB 1000
```

```
480 X = CX + J: Y = CY - J: GOSUB 1000
490 NEXT J
500 FOR J = 1 TO 2000: NEXT J: GOTO 200
1000 POKE 64*Y + X - 4033, A(0)
1010 FOR N = 0 TO INT(7 * J / 15)
1020 IF X <> CX THEN 1050
1030 Y2 = Y: X2 = X + N: GOSUB 2000
1040 X2 = X - N: GOSUB 2000: NEXT N: RETURN
1050 IF Y <> CY THEN 1080
1060 X2 = X: Y2 = Y + N: GOSUB 2000
1070 Y2 = Y - N: GOSUB 2000: NEXT N: RETURN
1080 Y2 = Y: IF X >= CX THEN 1100
1090 X2 = X + N: GOSUB 2000: GOTO 1110
1100 X2 = X - N: GOSUB 2000
1110 X2 = X: IF Y >= CY THEN 1130
1120 Y2 = Y + N: GOSUB 2000: GOTO 1140
1130 Y2 = Y - N: GOSUB 2000
1140 NEXT N: RETURN
2000 M = N: IF M > 6 THEN M = 6
2010 POKE 64*Y2 + X2 - 4033, A(M): RETURN
```

EASY CHANGES

1. Change line 300 to say D=−D instead of D=1. This will cause alternating inward and outward drawing of the designs rather than always outward from the center.
2. In line 500, change the constant of 2000 to 10000. This will cause a delay of about twenty seconds between the drawing of successive designs instead of three seconds. Or, remove the FOR-NEXT loop in line 500 to eliminate the delay entirely.
3. Modify the range of graphics characters from which the ones in the design are randomly selected. This is done by modifying the values of NC and T in line 160. For example, try

   ```
      160 T=128:NC=48
   or 160 T=142:NC=31
   or 160 T=177:NC=14
   ```

   Experiment with other values. Be sure that NC and T are both positive integers, that T is at least 128, and that the sum of NC and T is not greater than 191.
4. To permit a different character sequence, add:

   ```
   162 INPUT "Random value";R
   165 R=RND(−ABS(R))
   ```

MAIN ROUTINES

150-170	Initializes variables. Clears screen.
200	Picks seven random graphics characters.
300-310	Selects D, K, and L to draw patterns inward or outward.
400-490	Mainline routine. Determines axes of pattern.
500	Delays about three seconds after drawing design.
1000-1140	POKEs graphics character into axes of design, then determines coordinates of points between axes (subroutine).
2000-2010	POKEs characters between axes (subroutine).

MAIN VARIABLES

CX,CY	Coordinates of center of design. Upper left corner is considered to be (0,0).
D	Direction in which design is drawn (1=outward, −1=inward).
A	Array for the seven random graphics characters.
J,N,M	Subscript variables and loop counters.
T	Numeric representation of lowest graphics character to be used.
K,L	Inner and outer bounds of design (distance from center).
X,Y	Coordinates of character being POKEd on horizontal, vertical, or diagonal axis.
X2,Y2	Coordinates of character begin POKEd between horizontal, vertical, and diagonal axes.
NC	Number of possible graphics characters from which to choose.

SUGGESTED PROJECTS

1. Modify the program to use the entire 64 × 30 CRT screen, instead of just a 30 × 30 area in the center.
2. Add a routine to create truly symmetrical characters, using the Sorcerer's user-defined character capabilities.

SPARKLE

PURPOSE

This graphics display program provides a continuous series of hypnotic patterns, some of which seem to sparkle at you while they are being created. Two types of patterns are used. The first is a set of concentric diamond shapes in the center of the screen. Although the pattern is regular, the sequence in which it is created is random, which results in the "sparkle" effect.

The second type of pattern starts about two seconds after the first has finished. It is a series of "sweeps" across the screen–left to right and top to bottom. Each sweep uses a random graphics character that is spaced equally across the screen. The spacing distance is chosen at random for each sweep. Also, the number of sweeps to be made is chosen at random each time, from 10 to 30.

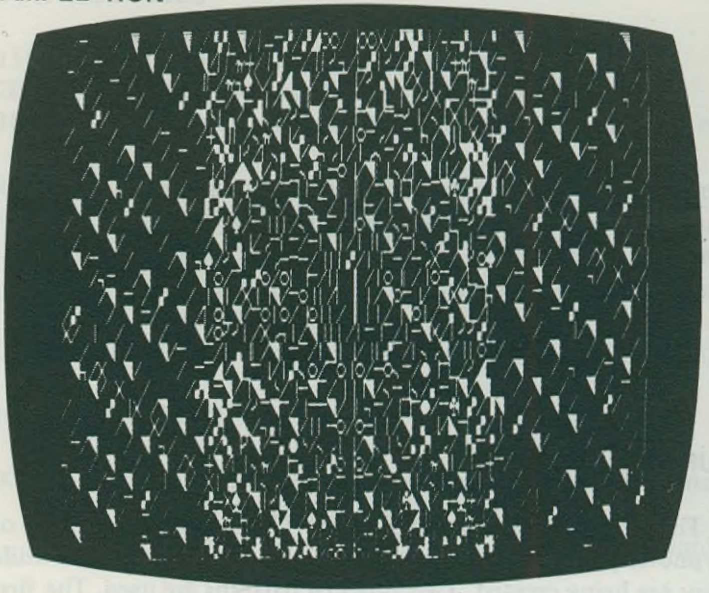
After the second type of pattern is complete, the program goes back to the first type, which begins to overlay the center of the screen again.

HOW TO USE IT

Confused by what you just read? Never mind. You have to see it to appreciate it. Just enter the program into your Sorcerer, then sit back and watch the results of your labor.

Hit **CTRL C** to stop at any time.

SAMPLE RUN



One of the "sparkling" patterns of SPARK.

PROGRAM LISTING

```
100 REM <SPARK>
110 REM Sparkling graphics display
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 DIM A(15), B(15)
160 CX = 32: CY = 15: T = 128: J = RND(-1)
170 PRINT CHR$(12)
200 FOR J = 0 TO 15: A(J) = J: B(J) = J: NEXT J
210 FOR J = 0 TO 15: R = INT(15 * RND(1))
220 W = A(J): A(J) = A(R): A(R) = W: NEXT J
230 FOR J = 0 TO 15: R = INT(15 * RND(1))
240 W = B(J): B(J) = B(R): B(R) = W: NEXT J
300 FOR J = 0 TO 15: FOR K = 0 TO 15
310 R = A(J): W = B(K)
320 C = T + INT(64 * RND(1) * (R + W) / 30)
330 X = CX + R: Y = CY + W: POKE 64*Y + X - 4033, C
340 Y = CY - W: POKE 64*Y + X - 4033, C
350 X = CX - R: POKE 64*Y + X - 4033, C
360 Y = CY + W: POKE 64*Y + X - 4033, C
```

```
370 X = CX + W: Y = CY + R: POKE 64*Y + X - 4033, C
380 Y = CY - R: POKE 64*Y + X - 4033, C
390 X = CX - W: POKE 64*Y + X - 4033, C
400 Y = CY + R: POKE 64*Y + X - 4033, C
410 NEXT K: NEXT J
500 FOR J = 1 TO 2000: NEXT J
600 FOR J = 1 TO INT(21 * RND(1)) + 10
610 R = 5 + INT(58*RND(1)): C = T + INT(63*RND(1))
620 FOR K=-4033 TO -2049 STEP R: POKE K, C: NEXT K
630 NEXT J: GOTO 210
```

EASY CHANGES

1. Make the second type of pattern appear first by inserting this line:

   205 GOTO 600

   Or, eliminate the first type of pattern by inserting:

   225 GOTO 600

   Or, eliminate the second type of pattern by inserting:

   420 GOTO 210
2. Increase the delay after the first type of pattern by increasing the 2000 in line 500 to, say, 5000. Remove line 500 to eliminate the delay.
3. Increase the number of sweeps of the second type of pattern across the screen by changing the 10 at the right end of line 600 into a 30 or a 50, for example. Decrease the number of sweeps by changing the 10 to a 1, and also changing the 21 in line 600 to 5 or 10.
4. Watch the effect on the second type of pattern if you change the 58 in line 610 to various integer values between 2 and 100.
5. Use only alphabetic and special characters by setting T=64 in line 160.

MAIN ROUTINES

150-170	Initializes variables. Clears screen.
200-410	Displays pattern in center of screen.
200-240	Shuffles the numbers 0 through 15 in the A and B arrays.
300-410	POKEs graphics characters to the screen.

500	Delays for about two seconds.
600-630	Overlays the entire screen with a random graphics character spaced at a fixed interval chosen at random.

MAIN VARIABLES

R	Random integer. Also, work variable.
A,B	Arrays in which shuffled integers from 0 to 15 are stored for use in making first type of pattern.
CX,CY	Coordinates of center of screen (32 across, 15 down).
T	Minimum ASCII value of characters.
J,K	Work and loop variables.
W	Work variable.
X,Y	Coordinates of a position on the screen for a graphics character, measured from the center.
C	Graphics character to be POKEd to screen at X,Y.

SUGGESTED PROJECT

Make the second type of pattern alternate between "falling from the top" (as it does now) and rising from the bottom of the screen.

SQUARES

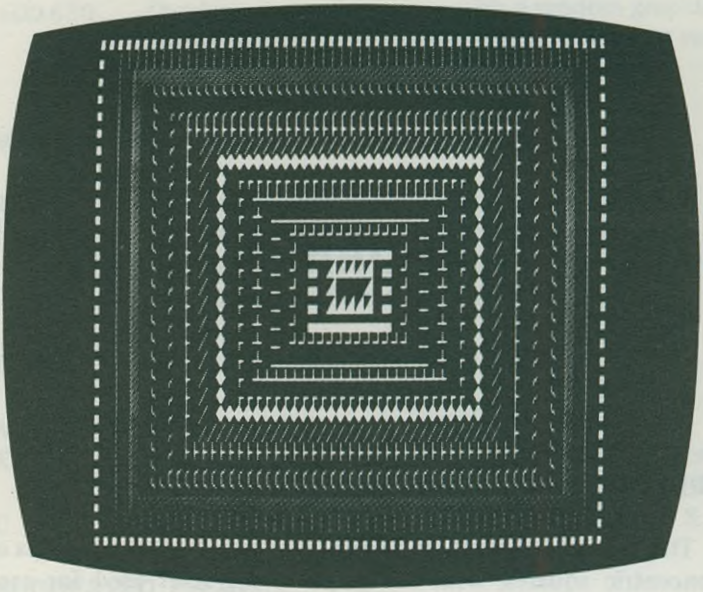
PURPOSE

This is another graphics-display program. It draws a series of concentric squares, with the graphics character used for each square chosen at random.

HOW TO USE IT

As with most of the other graphics display programs, you just sit back and enjoy watching this one once you get it started. Squares can be drawn at random, or in concentric order. Select your choice from the five available options shown in the menu. Hit **CTRL C** to end the program.

SAMPLE RUN



SQUAR creates a series of concentric graphics-character squares, which change in any of five selected manners.

PROGRAM LISTING

```
100 REM <SQUAR>
110 REM Graphic squares
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Russ & Phil Feldman
200 PRINT CHR$(12); "SQUARES": PRINT:PRINT
210 D = 1: J = RND(-1): PRINT "Options menu:"
220 PRINT "   Inward spiral -- enter '1'"
230 PRINT "   Outward spiral -- enter '2'"
240 PRINT "   Unidirectional, random -- enter '3'"
250 PRINT "   Bidirectional, random -- enter '4'"
260 PRINT "   In / out spiral -- enter '5'"
270 INPUT "Your choice"; R$: R = INT(VAL(R$))
280 IF R > 0 AND R < 6 THEN PRINT CHR$(12):GOTO 300
290 PRINT:PRINT:PRINT "What did you say?": GOTO 270
300 ON R GOTO 310, 330, 360, 350, 370
310 SN = SN - 1: IF SN < 0 THEN SN = 14
```

```
320 GOTO 390
330 SN = SN + 1: IF SN > 14 THEN SN = 0
340 GOTO 390
350 IF RND(1) > 0.5 THEN D = -D
360 SN = INT(14 * RND(1)): GOTO 390
370 IF D > 0 THEN 385
375 SN = SN - 1: IF SN < 0 THEN SN = 1: D = -D
380 GOTO 390
385 SN = SN + 1: IF SN > 14 THEN SN = 13: D = -D
390 C = 128 + INT(64 * RND(1))
400 TL = -3044 - 66 * SN: TR = TL + 4 * SN
410 BL = -3044 + 62 * SN: BR = BL + 4 * SN
420 IF D < 0 AND R <> 5 THEN 600
500 FOR J = TL TO TR: POKE J, C: NEXT J
510 FOR J = TR TO BR STEP 64: POKE J, C: NEXT J
520 FOR J = BR TO BL STEP -1: POKE J, C: NEXT J
530 FOR J = BL TO TL STEP -64: POKE J, C: NEXT J
540 GOTO 300
600 FOR J = TL TO BL STEP 64: POKE J, C: NEXT J
610 FOR J = BL TO BR: POKE J, C: NEXT J
620 FOR J = BR TO TR STEP -64: POKE J, C: NEXT J
630 FOR J = TR TO TL STEP -1: POKE J, C: NEXT J
640 GOTO 300
```

EASY CHANGE

To utilize alphabetics, change line 390 to:

```
390 C=65 + INT(26*RND(1))
```

MAIN ROUTINES

200-290	Initializes variables. Clears screen and gets user's input.
300-390	Selects square location and random character.
400-420	Finds POKE addresses for corners.
500-540	Draws a clockwise square.
600-640	Draws a counterclockwise square.

MAIN VARIABLES

J	Loop counter.
SN	Square number (0 = center, 14 = outer).

R,R$	User's choice of drawing options.
D	Direction flag.
C	ASCII value of output character.
TL,TR, BL,BR	POKE addresses of corners for square number SN.

SUGGESTED PROJECT

Modify the program to divide the screen into equal quarters, each with an independent set of squares.

ACROB

PURPOSE

The Sorcerer is quite a versatile machine. This program takes advantage of its powerful graphics capability to produce computer animation. That's right, animation! ACROB will entertain you with a presentation from the Sorcerer Stage Society.

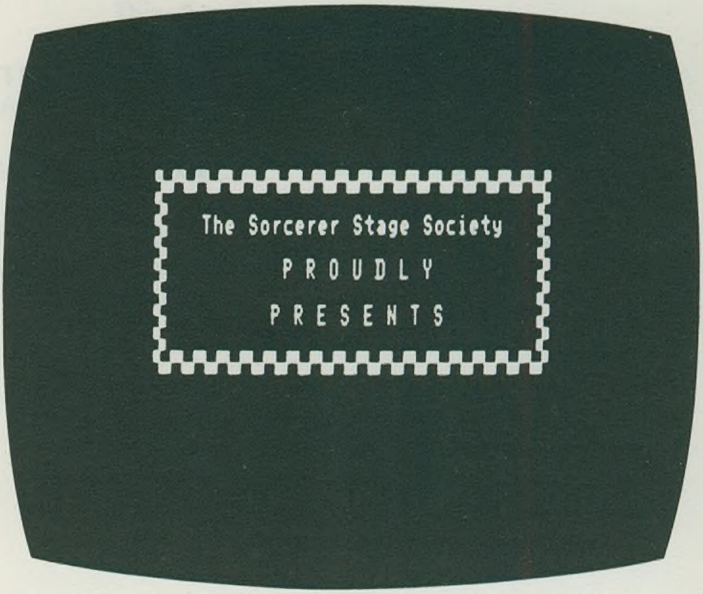
The Society searches the world over to bring you the best in circus acts and other performing artists. Today, direct from a performance before the uncrowned heads of Europe, the Society brings you the Flying Walloons.

HOW TO USE IT

Just sit back, relax, and get ready to enjoy the show. Type **RUN** and the Flying Walloons will be ready to perform. You have a front-row-center seat and the curtain is about to go up.

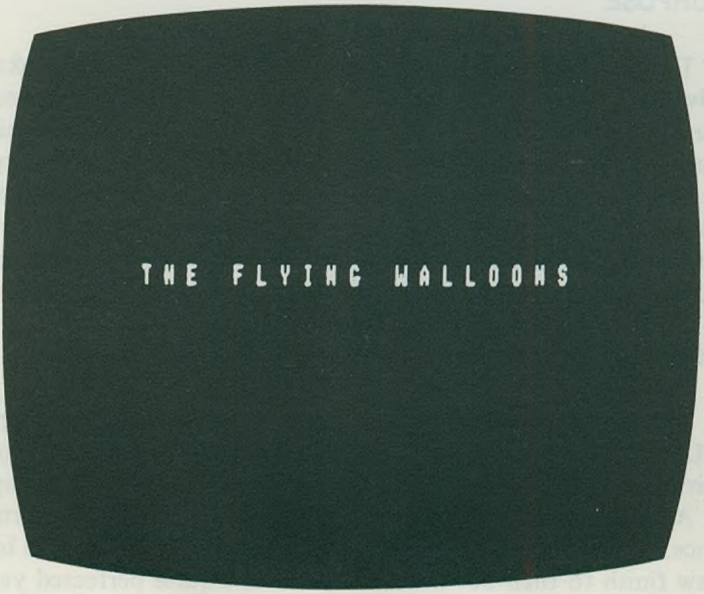
Applause might be appropriate if you enjoy their performance. Please note that the Walloons have been working on a big new finish to their act which they haven't quite perfected yet.

SAMPLE RUN



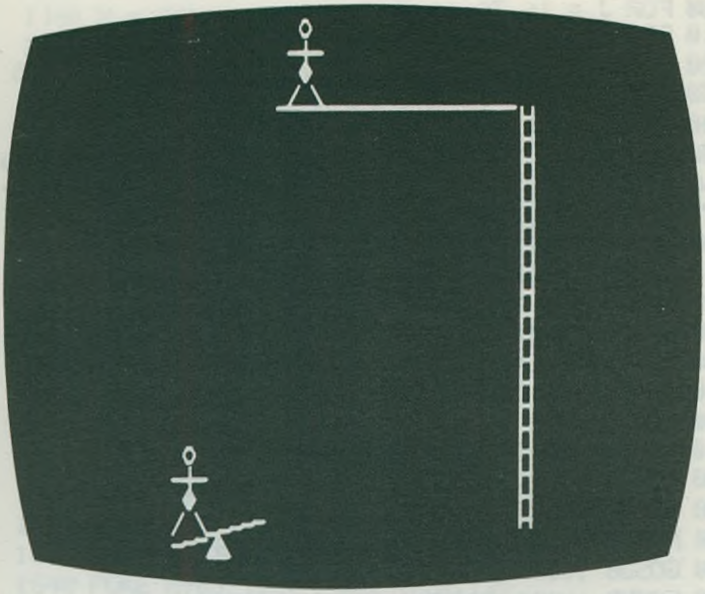
```
The Sorcerer Stage Society

       P R O U D L Y

      P R E S E N T S
```

The playbill announces a new presentation from the (in)famous Sorcerer Stage Society.



```
T H E   F L Y I N G   W A L L O O N S
```

Stand by for "The Flying Walloons!"



Watch the performance—if your heart can stand the excitement and suspense!

PROGRAM LISTING

```
100 REM <ACROB>
110 REM Animated acrobats
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 NJ = 4
160 GOTO 200
170 PRINT TAB(15); CHR$(165);: RETURN
180 PRINT TAB(48); CHR$(166): RETURN
200 GOSUB 1900: FOR J = 1 TO 10: PRINT: NEXT J
210 PRINT TAB(15);: FOR J = 16 TO 48 STEP 2
220 PRINT CHR$(165); CHR$(166);: NEXT J: PRINT
230 GOSUB 170: GOSUB 180: GOSUB 170
240 PRINT "   The Sorcerer Stage Society";
250 GOSUB 180: GOSUB 170: GOSUB 180: GOSUB 170
260 PRINT TAB(26); "P R O U D L Y";: GOSUB 180
270 GOSUB 170: GOSUB 180: GOSUB 170
280 PRINT TAB(25); "P R E S E N T S";: GOSUB 180
290 GOSUB 170: GOSUB 180: PRINT TAB(15);
```

```
300 FOR J = 16 TO 48 STEP 2
310 PRINT CHR$(165); CHR$(166);: NEXT J: PRINT
320 GOSUB 1930: PRINT CHR$(17);: FOR J = 1 TO 15
330 PRINT: NEXT J: GOSUB 170: PRINT SPC(30);
340 GOSUB 180: GOSUB 1920: GOSUB 1900
350 FOR J = 1 TO 14: PRINT: NEXT J: PRINT TAB(13);
360 PRINT "T H E   F L Y I N G   W A L L O O N S"
370 GOSUB 1930: GOSUB 1900
500 GOSUB 1700: GOSUB 1950: Y = 21: X = 12
510 GOSUB 1800: X = 62: GOSUB 1800: GOSUB 1920
520 FOR J = 0 TO 18: X = J + 11: Y = 21
530 GOSUB 2000: GOSUB 1850: X = X + 1: GOSUB 1800
540 X = 62: Y = 21 - J: GOSUB 2000: GOSUB 1850
550 Y = Y - 1: GOSUB 1800: NEXT J: GOSUB 1850
600 FOR X = 59 TO 39 STEP -1
610 GOSUB 1800: GOSUB 1850: NEXT X
620 FOR X = 39 TO 50: GOSUB 1800: GOSUB 1910
630 GOSUB 1850: NEXT X
640 FOR X = 50 TO 38 STEP -2: GOSUB 1800
650 GOSUB 1850: NEXT X: X = 37: Y = 3
660 GOSUB 1800: GOSUB 1850: X = 36: Y = 4
670 GOSUB 1800: GOSUB 1850: R = 0: GOTO 800
800 X = 35: FOR Y = 4 TO 6: GOSUB 1800
810 GOSUB 1850: NEXT Y: FOR Y = 8 TO 12 STEP 2
820 GOSUB 1800: GOSUB 1850: NEXT Y
830 FOR Y = 15 TO 21 STEP 3: GOSUB 1800
840 GOSUB 1850: NEXT Y: Y = 21: GOSUB 1800:
850 X = 30: GOSUB 2000: GOSUB 1850:GOSUB 1600
860 X = 30: FOR Y = 19 TO 13 STEP -3: GOSUB 1800
870 GOSUB 1850: NEXT Y: FOR Y = 13 TO 5 STEP -2
880 GOSUB 1800: GOSUB 1850: NEXT Y
890 FOR Y = 4 TO 1 STEP -1: GOSUB 1800
900 GOSUB 1850: NEXT Y: FOR Y = 1 TO 4
910 GOSUB 1800: GOSUB 1850: NEXT Y
920 FOR Y = 6 TO 14 STEP 2: GOSUB 1800
930 GOSUB 1850: NEXT Y: FOR Y = 17 TO 20 STEP 3
940 GOSUB 1800: GOSUB 1850: NEXT Y
950 X = 35: Y = 21: GOSUB 2000: GOSUB 1850
960 GOSUB 1700: X = 30: Y = 21: GOSUB 1800: X = 35
970 FOR Y = 18 TO 12 STEP -3: GOSUB 1800
980 GOSUB 1850: NEXT Y: FOR Y = 10 TO 6 STEP -2
990 GOSUB 1800: GOSUB 1850: NEXT Y
1000 FOR Y = 5 TO 3 STEP -1: GOSUB 1800
1010 GOSUB 1850: NEXT Y
1020 R = R + 1: IF R <= NJ THEN 800
```

```
1100 X = 36: FOR Y = 4 TO 6: GOSUB 1800
1110 GOSUB 1850: NEXT Y: X = 37
1120 FOR Y = 8 TO 12 STEP 2: GOSUB 1800
1130 GOSUB 1850: NEXT Y: X = 38
1140 FOR Y = 15 TO 21 STEP 3: GOSUB 1800
1150 GOSUB 1850: NEXT Y: Y = 22: GOSUB 1800
1160 GOSUB 1850: Y = 24: X = 39: GOSUB 2000
1170 POKE A,151: POKE A+1,151: POKE A+2,144
1180 POKE A+3,143: POKE A+4,171
1190 POKE A+64,143: POKE A+65,151
1200 POKE A+66,163: POKE A+67,178
1210 POKE A+68,136
1220 FOR J = 1 TO 7: PRINT: NEXT J
1230 GOSUB 1930: PRINT TAB(26); "F I N I S"
1240 GOSUB 1930: GOSUB 1900: END
1600 X = 29: Y = 24: GOSUB 2000
1610 POKE A,140: POKE A+1,151: POKE A+2,174
1620 POKE A+3,175: POKE A+4,176
1630 POKE A+67,159: POKE A+68,158
1640 POKE A+69,138: POKE A+70,139
1650 POKE A+71, 140: POKE A+6,32
1660 POKE A+64, 32: POKE A+65,32
1670 POKE A+66, 32: RETURN
1700 X = 29: Y = 25: GOSUB 2000
1710 POKE A,140: POKE A+1,139: POKE A+2,138
1720 POKE A+3,159: POKE A+4,158
1730 POKE A-61,176: POKE A-60,175
1740 POKE A-59,174: POKE A-58,151
1750 POKE A-57,140: POKE A-63,32
1760 POKE A+5, 32: POKE A+6, 32
1770 POKE A+7,32: RETURN
1800 GOSUB 2000: POKE A,136: POKE A+63,151
1810 POKE A+64,173: POKE A+65,151
1820 POKE A+128,163: POKE A+191,171
1830 POKE A+193,172: RETURN
1850 POKE A,32: POKE A+63,32: POKE A+64,32
1860 POKE A+65,32: POKE A+128,32: POKE A+191,32
1870 POKE A+193,32: RETURN
1900 PRINT CHR$(12): RETURN
1910 FOR L = 1 TO 100: NEXT L: RETURN
1920 FOR L = 1 TO 1000: NEXT L: RETURN
1930 FOR L = 1 TO 2000: NEXT L: RETURN
1950 Y = 6: FOR X = 38 TO 58: GOSUB 2000
1960 POKE A,137: NEXT X
```

```
1970 X = 59: FOR Y = 6 TO 24: GOSUB 2000
1980 POKE A,180: POKE A+1,185: NEXT Y: RETURN
2000 A = 64*Y + X - 4033: RETURN
```

EASY CHANGES

1. If you wish to have the Walloons perform more (or less) jumps during their performance, change the value of NJ in line 150 accordingly. To get five full jumps, use

   150 NJ=5

2. Timing delays are used often in the program. To change the length of the delay, alter the values in lines 1910-1930. Larger values will lengthen the delays, while smaller ones will shorten the delays.
3. You might want to personalize the title placard and make yourself the presenter of the Walloons. This can be done by altering the string literal in line 240 to something else. However, you cannot use a string with a length of much more than 32 characters or it will print beyond the end of the placard. To say, for example, that Simon Fenster presents the Walloons, change line 240 to:

   240 PRINT SPC(9); "Simon Fenster";

MAIN ROUTINES

150-180	Sets jump counter, playbill.
200-310	Displays title placard.
320-370	Removes "proudly," displays rest of title.
500-550	Moves first Walloon into view.
600-670	Second Walloon enters from the high platform.
800-1020	Flying Walloons perform.
1100-1240	Concludes Walloons' performance.
1600-1670	Subroutine to draw lever with right side down.
1700-1770	Subroutine to draw lever with right side up.
1800-1830	Subroutine to draw Walloon with head at POKE location A.
1850-1870	Subroutine to erase Walloon with head at A.
1900	Subroutine to clear screen.
1910-1930	Time-delaying subroutines.
1950-1980	Subroutine to draw ground and ladder.

2000	Subroutine to calculate A from X,Y position.

MAIN VARIABLES

NJ	Number of jumps to make.
A	Reference POKE location.
R	Jump counter.
J,K,L	Loop counters.
X,Y	Column and line location of head (from top left corner).

SUGGESTED PROJECTS

1. There are many possibilities for "spicing up" the Walloons' act with extra tricks or improved ones. Perhaps you would like to change their finish to something less crude.
2. If you add some alternate tricks or endings as suggested in the previous project, try randomizing if and when they will be done. Thus, the Walloons' performance will be different each time the program is run. At least their ending may be variable.
3. Scour the world yourself for acts. Maybe someday we will have a complete software library of performing artists—including some competent ones.

Section 5

Mathematics Programs

INTRODUCTION TO MATHEMATICS PROGRAMS

Since their invention, computers have been used to solve mathematical problems. Their great speed and reliability render solvable many otherwise difficult (or impossible) calculations. Dozens of numerical techniques lend themselves naturally to computer solution. The following programs explore some of them. They will be of interest mainly to engineers, students, mathematicians, statisticians, and others who encounter such problems in their work.

GRAPH takes advantage of the Sorcerer's user-defined characters to draw the graph of a function Y = f(X). The function is supplied by you. INTEG calculates the integral, or "area under the curve," for any such function.

Experimental scientific work frequently results in data at discrete values of X and Y. CURVE finds a polynomial algebraic expression to express this data as a formula.

Theoretical scientists (and algebra students) often must find the solution to a set of simultaneous linear algebraic equations. SIMEQ does the trick.

Much modern engineering work requires the solution of differential equations. DIFEQ will solve any first-order ordinary differential equation that you provide.

STATS will take a list of data and derive standard statistical information describing it. In addition, it will sort the data list into ranking numerical order.

CURVE

PURPOSE

CURVE fits a polynomial function to a set of data. The data must be in the form of pairs of X-Y points. This type of data frequently occurs as the result of some experiment, or perhaps from sampling tabular data in a reference book.

There are many reasons why you might want an analytic formula to express the functional relationship inherent in the data. You will often have experimental errors in the Y values. A good formula expression tends to smooth out these fluctuations. Perhaps you want to know the value of Y at some X which is not obtained exactly in the experiment. This may be a point between known X values (interpolation) or one slightly outside the experimental range (extrapolation). If you wish to use the data in a computer program, a good formula is a convenient and efficient way to do it.

This program fits a curve of the form

$$Y = C_0 + C_1X^1 + C_2X^2 + \ldots + C_nX^n$$

to your data. You may select n, the power of the highest term, to be as large as 18. The constant coefficients, C_0 - C_n, are the main output of the program. Also calculated is the goodness of fit, a guide to the accuracy of the equation. You may fit different degree polynomials to the same data and also ask to have Y calculated for specific values of X.

The numerical technique involved in the computation is known as least squares curve fitting. It minimizes the sum of the squares of the errors. The least squares method reduces the problem to a set of simultaneous algebraic equations. Thus these equations could be solved by the algorithm used in SIMEQ. In fact, once the proper equations are set up, CURVE uses the identical subroutine found in SIMEQ to solve the equations. For more information, the bibliography contains references to descriptions of the numerical technique.

HOW TO USE IT

The first thing you must do, of course, is enter the data into the program. This consists of typing in pairs of numbers. Each pair represents an X value and its corresponding Y value. The two numbers of each pair are input separately. A question mark will prompt you for each data item. After you have entered them all, type **END** to signal the end of the data. When you do this, the program will respond by indicating how many data pairs have been entered. A maximum of 100 data pairs is allowed.

Next, you must input the degree of the polynomial to be fitted. FIT can be any nonnegative integer subject to certain constraints. The maximum allowed is 18. Unless your data is well-behaved (X and Y values close to 1), the program will often produce inaccurate results if FIT is greater than 5 or so. This is because sums of powers of X and Y are calculated up to powers of 2*FIT. These various sums differ from each other by several orders of magnitude. Errors result because of the numerous truncation and round-off operations involved in doing arithmetic with them. Also, FIT must be less than the number of data pairs. You will get an error message if you input an illegal value of FIT.

A few notes regarding the selection of FIT may be of interest. If FIT=0, the program will output the mean value of Y as the coefficient C_0. If FIT=1, the program will be calculating the best straight line through the data. This special case is known as "linear regression." If FIT is one less than the number of data pairs, the program will find an exact fit to the data (barring round-off and other numerical errors). This is a solution which passes exactly through each data point.

Once you have entered the desired degree, the program will begin calculating the results. There will be a pause while this calculation is performed. The time involved depends on the number of data pairs and the degree selected. For 25 data pairs and a third degree fit, the pause will be about 15 seconds. If 50 data pairs and a fifth degree fit is specified, the pause will be about 45 seconds.

The output gives the values of the coefficients for each power of X from 0 to FIT. Also shown is the percent goodness of fit. This is a measure of how accurately the program was able to construct the curve in a given case. A value of 100 percent means perfect fit, lesser values indicate correspondingly poorer fits. It is hard to say what value denotes *satisfactory* fit since much depends on the accuracy of data and the purpose at hand. But, as a rule of thumb, anything in the high nineties is quite good. For those interested, the formula to calculate the percent goodness of fit is

$$\mathrm{PGF} = 100 * \sqrt{1 - [\sum_i (Y_i - \hat{Y}_i)^2 / \sum_i (Y_i - \bar{Y})^2]}$$

where Y_i are the actual Y data values, $\hat{Y}_i$ are the calculated Y values (through the polynomial expression), and $\bar{Y}$ is the mean value of Y.

Next, you are presented with four options for continuing the run. These are: 1) determining specific points, 2) fitting another degree, 3) restarting, and 4) ending the program. Simply input **1**, **2**, **3**, or **4** to make your selection.

Option 1 allows you to see the value of Y that the current fit will produce for a given value of X. In this mode you are continually prompted to supply any value of X. The program then shows what the polynomial expression produces as the value for Y. Input **END** for an X value to leave this mode.

Option 2 allows you to fit another degree polynomial to the same data. Frequently, you will want to try successively higher values of FIT to improve the goodness. Unless round-off errors occur, this will cause the percent goodness of fit to increase.

Option 4 simply terminates the program, while option 3 restarts it for new data.

SAMPLE PROBLEM

Problem: An art investor is considering the purchase of Finfrock's masterpiece, "Fundamental Fantasy." Since 1940, the painting has been for sale at auction seven times. Here is the painting's sales record from these auctions:

Year	*Price*
1940	$ 8000
1948	$13000
1951	$16000
1956	$20000
1962	$28000
1968	$39000
1975	$53000

The painting is going to be sold at auction in 1979. What price should the investor expect to have to pay to purchase the painting? If he resold it in 1983, how much profit should he expect to make?

Solution: The investor will try to get a polynomial function that expresses the value of the painting as a function of the year, using CURVE. The year will be represented by the variable X, and the price is shown by the variable Y. To keep the magnitude of the numbers small, the years will be expressed as elapsed years since 1900, and the price will be in units of $1000. (Thus a year of 40 represents 1940, and a price of 8 represents $8000.)

SAMPLE RUN

```
LEAST SQUARES CURVE-FITTING ROUTINE

Enter a data value in response to each request.
The first value input is an X value; the
second is the associated Y value.

Input of X-Y pairs continues until 'END' is
entered in response to an input request. Up to
100 data pairs can be input.


Data pair # 1
X value? 40
Y value? 8

Data pair # 2
X value ? 48
Y value? 13

Data pair # 3
X value? 51
Y value? 16
```

```
Data pair # 4
X value? 56
Y value? 20

Data pair # 5
X value? 62
Y value? 28

Data pair # 6
X value? 68
Y value? 39

Data pair # 7
X value? 75
Y value? 53

Data pair # 8
X value? END
A total of 7 data pairs entered.

Desired degree of polynomial? 1

Y = f(x) = -48.2707* X↑0 + 1.28724 *X↑1

Goodness of fit = 97.5302 %

Options menu:
   Determine specific points -- enter '1'
   Fit another degree -- enter '2'
   Start over -- enter '3'
   End program -- enter '4'
Your choice? 2

Desired degree of polynomial? 2

Y = f(x) = +38.5297 *X↑0 - 1.83686 *X↑1
              +.0270514 *X↑2

Goodness of fit = 99.9486 %
   :
(Options menu displayed again)
   :
Your choice? 1

X value? 79
Y value = 62.2451

X value? 83
Y value = 72.427

X value? END
   :
(Options menu displayed again)
   :
Your choice? 4
```

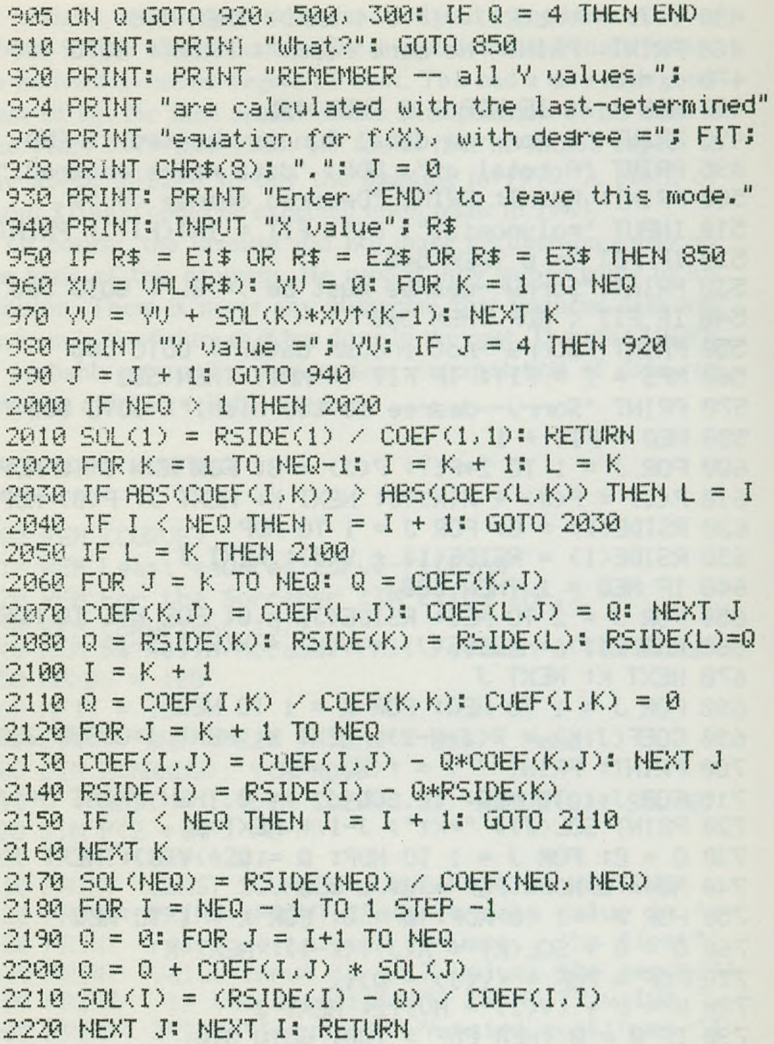
Initially, a first degree fit was tried and a goodness of fit of about 97.5% was obtained. The investor wanted to do better, so he tried a second degree fit next. This had a very high goodness of fit. He then asked for the extrapolation of his data to the years 1979 and 1983. He found that he should expect to pay about $62250 to buy the painting in 1979. A profit of about $10000 could be expected upon resale in 1983.

Of course, the investor did not make his decision solely on the basis of this program. He used it only as one guide to his decision. There is never any guarantee that financial data will perform in the future as it has done in the past. Though CURVE is probably as good a way as any, extrapolation of data can never be a totally reliable process.

PROGRAM LISTING

```
100 REM <CURVE>
110 REM Least squares curve-fitting
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 MAXDV = 100
160 MFIT = 18
170 E1$ = "END": E2$ = "End": E3$ = "end"
200 DIM X(MAXDV), Y(MAXDV)
210 Q = MFIT + 1: DIM COEF(Q,Q), RSIDE(Q), SOL(Q)
220 DIM P(2 * ~~MF~~) MFIT
300 PRINT CHR$(12);
310 PRINT "LEAST SQUARES CURVE-FITTING ROUTINE"
320 PRINT: PRINT: PRINT "Enter a data value in ";
330 PRINT "response to each request.  The first"
340 PRINT "value input is an X value; the second";
350 PRINT " is the associated": PRINT "Y value."
360 PRINT: PRINT "Input of X-Y pairs continues ";
370 PRINT "until 'END' is entered in response"
380 PRINT "to an input request.  Up to"; MAXDV;
390 PRINT "data pairs can be input."
400 PRINT: PRINT: J = 0
410 J = J + 1: PRINT: IF J/4 <> INT(J/4) THEN 420
415 PRINT: PRINT "Enter 'END' to stop input mode."
420 PRINT: PRINT "Data pair #"; J
425 INPUT "X value"; R$
430 IF R$ = E1$ OR R$ = E2$ OR R$ = E3$ THEN 470
440 X(J) = VAL(R$): INPUT "Y value"; R$
```

```
450 Y(J) = VAL(R$): IF J < MAXDV THEN 410
460 PRINT: PRINT "No more room!": PRINT: GOTO 480
470 J = J - 1
480 NDP = J: IF NDP > 0 THEN 495
490 PRINT "Error--no data! Run is aborted.": END
495 PRINT "A total of"; NDP; "data pairs entered."
500 PRINT: PRINT: PRINT "Desired degree of ";
510 INPUT "polynomial"; FIT: FIT = INT(FIT): PRINT
520 IF FIT >= 0 THEN 540
530 PRINT "Sorry--degree must be >= 0.": GOTO 500
540 IF FIT < NDP THEN 560
550 PRINT "Sorry--not enough data.": GOTO 500
560 MPS = 2 * FIT: IF FIT <= MFIT THEN 580
570 PRINT "Sorry--degree is too high.": GOTO 500
580 NEQ = FIT + 1
600 FOR J = 1 TO 2*FIT: P(J) = 0: FOR K = 1 TO NDP
610 P(J) = P(J) + X(K)↑J: NEXT K: NEXT J: P(0)=NDP
620 RSIDE(1) = 0: FOR J = 1 TO NDP
630 RSIDE(1) = RSIDE(1) + Y(J): NEXT J
640 IF NEQ = 1 THEN 680
650 FOR J = 2 TO NEQ: RSIDE(J) = 0: FOR K=1 TO NDP
660 RSIDE(J) = RSIDE(J) + Y(K) * X(K)↑(J-1)
670 NEXT K: NEXT J
680 FOR J = 1 TO NEQ: FOR K = 1 TO NEQ
690 COEF(J,K) = P(J+K-2): NEXT K: NEXT J:GOSUB 2000
700 PRINT: PRINT " Y = f(X) =",
710 FOR J=1 TO NEQ: IF SOL(J) >= 0 THEN PRINT "+";
720 PRINT SOL(J); "*X↑"; J-1,: NEXT J
730 Q = 0: FOR J = 1 TO NDP: Q = Q + Y(J): NEXT J
740 MV = Q/NDP: PGF = 0: W = 0
750 FOR J = 1 TO NDP: Q = 0: FOR K = 1 TO NEQ
760 Q = Q + SOL(K) * X(J)↑(K-1): NEXT K
770 PGF = PGF + (Y(J) - Q)↑2
780 W = W + (Y(J) - MV)↑2: NEXT J
790 IF W = 0 THEN PGF = 100: GOTO 810
800 PGF = 100 * SQR(1 - PGF/W)
810 PRINT: PRINT: PRINT "Goodness of fit =";
820 PRINT PGF; "%"
850 PRINT: PRINT "Options menu:"
860 PRINT "   Determine specific points -- ";
870 PRINT "enter '1'": PRINT "   Fit another ";
880 PRINT "degree to same data -- enter '2'"
885 PRINT "   Start over -- enter '3'"
890 PRINT "   End program -- enter '4'"
900 INPUT "Your choice"; R$: Q = VAL(R$)
```



```
905 ON Q GOTO 920, 500, 300: IF Q = 4 THEN END
910 PRINT: PRINT "What?": GOTO 850
920 PRINT: PRINT "REMEMBER -- all Y values ";
924 PRINT "are calculated with the last-determined"
926 PRINT "equation for f(X), with degree ="; FIT;
928 PRINT CHR$(8); ".": J = 0
930 PRINT: PRINT "Enter 'END' to leave this mode."
940 PRINT: INPUT "X value"; R$
950 IF R$ = E1$ OR R$ = E2$ OR R$ = E3$ THEN 850
960 XV = VAL(R$): YV = 0: FOR K = 1 TO NEQ
970 YV = YV + SOL(K)*XV↑(K-1): NEXT K
980 PRINT "Y value ="; YV: IF J = 4 THEN 920
990 J = J + 1: GOTO 940
2000 IF NEQ > 1 THEN 2020
2010 SOL(1) = RSIDE(1) / COEF(1,1): RETURN
2020 FOR K = 1 TO NEQ-1: I = K + 1: L = K
2030 IF ABS(COEF(I,K)) > ABS(COEF(L,K)) THEN L = I
2040 IF I < NEQ THEN I = I + 1: GOTO 2030
2050 IF L = K THEN 2100
2060 FOR J = K TO NEQ: Q = COEF(K,J)
2070 COEF(K,J) = COEF(L,J): COEF(L,J) = Q: NEXT J
2080 Q = RSIDE(K): RSIDE(K) = RSIDE(L): RSIDE(L)=Q
2100 I = K + 1
2110 Q = COEF(I,K) / COEF(K,k): CUEF(I,K) = 0
2120 FOR J = K + 1 TO NEQ
2130 COEF(I,J) = CUEF(I,J) - Q*COEF(K,J): NEXT J
2140 RSIDE(I) = RSIDE(I) - Q*RSIDE(K)
2150 IF I < NEQ THEN I = I + 1: GOTO 2110
2160 NEXT K
2170 SOL(NEQ) = RSIDE(NEQ) / COEF(NEQ, NEQ)
2180 FOR I = NEQ - 1 TO 1 STEP -1
2190 Q = 0: FOR J = I+1 TO NEQ
2200 Q = Q + COEF(I,J) * SOL(J)
2210 SOL(I) = (RSIDE(I) - Q) / COEF(I,I)
2220 NEXT J: NEXT I: RETURN
```

EASY CHANGES

1. The program uses **END** as the flag to terminate various inputs. You can easily change the flag by modifying the strings assigned to E1$, E2$, and E3$ in line 170. To use **XXX**, for example, make this change:

   170 E1$ = “XXX”: E2$ = “xxx”: E3$ = “Xxx”

2. Currently, a maximum of 100 data pairs is allowed. If you need more, change the value of MAXDV in line 150. To use up to 200 pairs, for example, use this line:

```
150 MAXDV = 200
```

MAIN ROUTINES

150-170	Initializes constants.
200-220	Dimensions arrays.
300-390	Displays introductory messages.
400-460	Gets X-Y input data from the user.
470-580	Gets degree of polynomial from the user, determines if it is acceptable.
600-720	Sets up equations for the simultaneous-equation solver and calls it.
730-790	Calculates percent goodness of fit, displays all results.
800-910	Gets user's continuation option and branches to it.
920-990	Determines Y value corresponding to any X value.
2000-2220	Subroutine to solve simultaneous linear algebraic equations.

MAIN VARIABLES

MAXDV	Maximum number of data pairs allowed.
MFIT	Maximum degree allowed to fit.
E1$,E2$,E3$	Ending flags for data input and X point mode.
X,Y	Arrays of X and Y data points.
NDP	Number of data pairs entered.
FIT	Degree of polynomial to fit.
MPS	2*FIT, the maximum power sum to compute.
NEQ	FIT+1, number of simultaneous equations to solve.
COEF, SOL, RSIDE	Arrays for simultaneous linear equation solver.
P	Array for holding sums of various powers of X.
I,J,K,L	Loop indices and counters.
Q,W	Work variables.
MV	Mean value of Y.
PGF	Percent goodness of fit.

R$	Input value.
XV	Specific X point for which to calculate Y.
YV	Y value corresponding to XV.

SUGGESTED PROJECTS

1. No provision for modifying the data is incorporated into the program. Often it would be nice to add, subtract, or modify parts of the data after some results are seen. Build in a capability to do this.
2. You may desire other forms of output. A useful table for many applications might include the actual X and Y values, calculated Y values and percentage errors in Y.
3. Sometimes certain points (or certain regions of points) are known to be more accurate than others. Then you would like to weight these points as being more important than others to be fit correctly. The least squares method can be modified to include such a weighting parameter with each data pair. Research this technique and incorporate it into the program. (Note: You can achieve some weighting with the current program by entering important points two or more times. There is a certain danger in this, however. You must only ask for a solution with FIT less than the number of *unique* data points. A division by zero error may result otherwise.)
4. Often you wish to try successively higher degree polynomials until a certain minimum percent goodness of fit is obtained. Modify the program to accept a minimally satisfactory percent goodness of fit from the user. Then have the program automatically try various polynomial fits until it finds the lowest degree fit, if any, with a satisfactory goodness of fit.

NOTES :-
THE DATA DOES NOT HAVE TO BE IN ANY PARTICULAR ORDER. ie. THE DATA PAIRS X/Y HAVE TO BE IN THE ORDER X THEN Y BUT THE PAIRS OF X/Y VALUES CAN BE INPUT IN ANY ORDER.

DIFFEQN

PURPOSE

Differential equations express functions by giving the rate of change of one variable with respect to another. This type of relation occurs regularly in almost all the physical sciences. The solution of these equations is necessary in many practical engineering problems.

For many such equations, a closed form (or exact analytical expression) solution can be obtained. However, for just as many, no such "simple" solution exists. The equation must then be solved numerically, usually by a computer program such as this.

There are many types and classes of differential equations. This program solves those of a simple type, namely, first order, ordinary differential equations. This means that the equation to be solved can be written in the form

$$\frac{dY}{dX} = \text{(any function of X,Y)}$$

Here, X is the independent variable and Y is the dependent variable. The equation expresses the derivative (or rate of change) of Y with respect to X. The right-hand side is an expression which may involve X and/or Y.

To use the program, you must supply it with the differential equation to be solved. The procedure to do this is explained in the "How To Use It" section.

A technique known as the "fourth-order, Runge-Kutta" method is used to solve the equation. Space limitations prevent any detailed explanation of it here. However, it is thoroughly discussed in the numerical analysis books referenced in the bibliography.

The program allows two forms of output. You can have the answers tabulated in columns or plotted graphically.

HOW TO USE IT

The first thing you must do is enter the differential equation into the program. This must be done at line 3000. Currently this line contains a PRINT statement. This will cause a warning message to be displayed if the program is run before you have changed line 3000. The form of line 3000 should be:

```
3000 D = (your function of X,Y)
```

D represents dY/dX. GOSUBs are made to line 3000 with X and Y set to their current values. Thus, when each RETURN is made, D will be set to the appropriate value of dY/dX for that given X and Y. If necessary, you may use the lines between 3000 and 3999 to complete the definition of D. Line 3999 already contains a RETURN statement so you do not need to add another one.

The program begins by warning you that you should have already entered the equation at line 3000. You acknowledge that this has been done by entering C to continue.

Now the various initial conditions are input. You are prompted for them one at a time. They consist of the initial values of X and Y, the stepsize interval in X at which to display the output, and the final value of X.

You now have a choice between two types of output. Enter a T for tabular output or a G for graphical output. The tabular form is simply a two-column display of the corresponding values of X and Y.

The graphical output plots the values of Y along a horizontal axis as each corresponding X value is displayed on successive lines of the screen. This graphical display requires you to input the minimum and maximum values of Y that will be used on the Y axis. You will be prompted for them if this output form is chosen. An open circle is used to plot the value of Y. If, however, the value of Y is "off-scale," a closed circle is plotted at the appropriate edge of the graph.

With the input phase completed, the program begins the selected output. The output is displayed at each interval of the stepsize until the final value of X is reached, 20 lines at a time. Hit **RETURN** for each new screen, for either the tabular or graphical form of output.

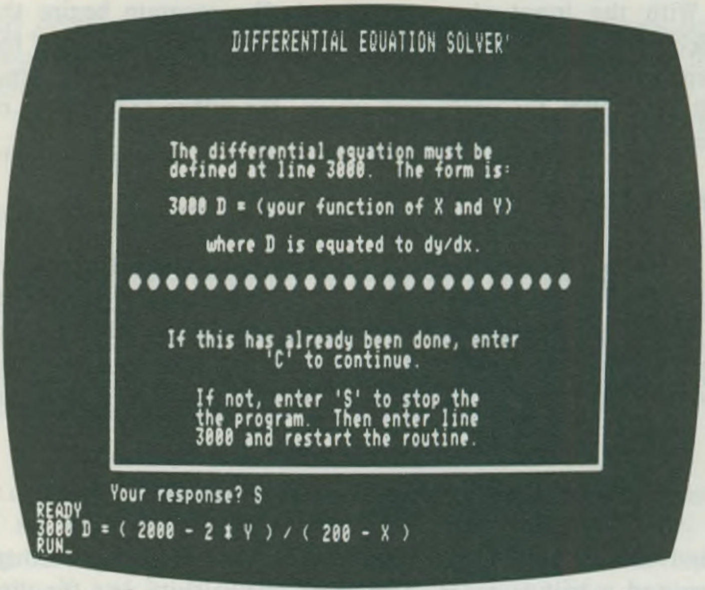
SAMPLE RUN

Problem: A body, originally at rest, is subjected to a force of 2000 dynes. Its initial mass is 200 grams. However, while it moves, it loses mass at the rate of 1 gram/sec. There is also an air resistance equal to twice its velocity retarding its movement. The differential equation expressing this motion is:

$$\frac{dY}{dX} = \frac{(2000 - 2Y)}{(200 - X)} \quad \begin{array}{l}\text{where } Y = \text{velocity (cm/sec)} \\ \qquad\quad X = \text{time (sec)}\end{array}$$

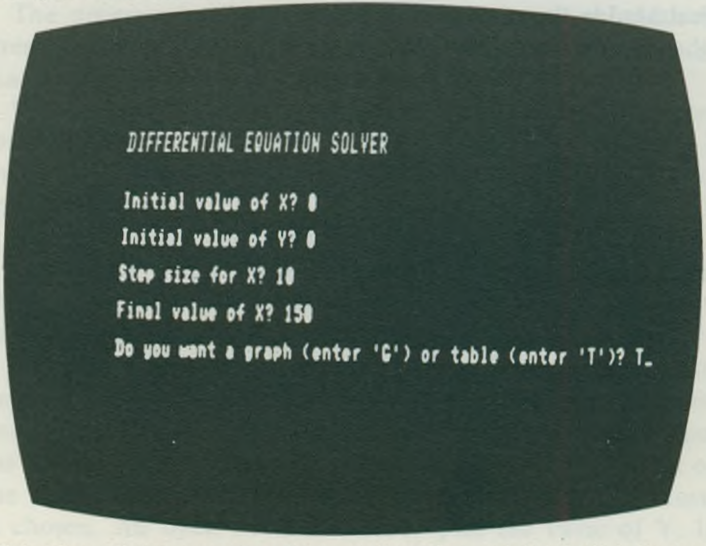
Find the velocity of the body every 10 seconds up through two and a half minutes. Also, plot this velocity as a function of time.

Solution and Sample Run: The solution and sample run are illustrated in the accompanying photographs.



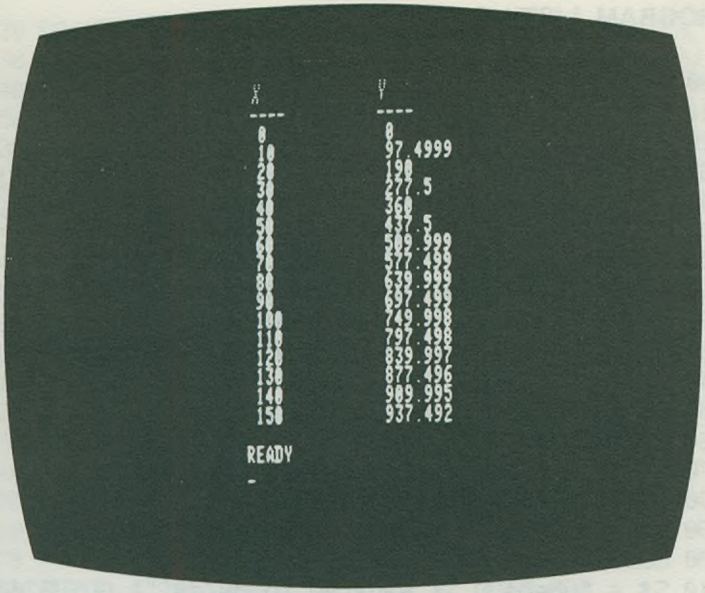
```
                DIFFERENTIAL EQUATION SOLVER'

          The differential equation must be
          defined at line 3000.  The form is:

          3000 D = (your function of X and Y)

              where D is equated to dy/dx.

     ●●●●●●●●●●●●●●●●●●●●●●●

          If this has already been done, enter
                    'C' to continue.

              If not, enter 'S' to stop the
              the program.  Then enter line
              3000 and restart the routine.

     Your response? S
READY
3000 D = ( 2000 - 2 * Y ) / ( 200 - X )
RUN_
```

Line 3000 of the program must contain the definition of the differential equation. Since it hasn't been entered, the user exits the program after the initial display and types in a new line 3000. The program is restarted using the RUN command.



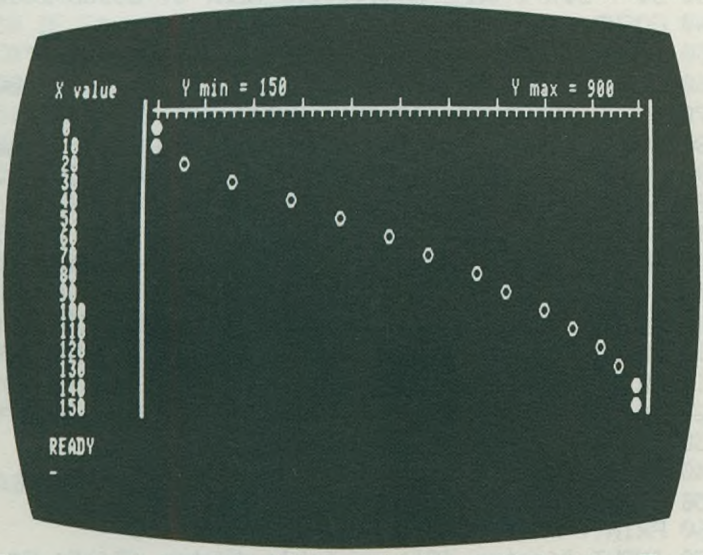
```
DIFFERENTIAL EQUATION SOLVER

Initial value of X? 0

Initial value of Y? 0

Step size for X? 10

Final value of X? 150

Do you want a graph (enter 'G') or table (enter 'T')? T_
```

This time, the user enters **C** to continue beyond the intiial display. Initial values for X and Y, along with incremental and maximum X values, are selected for a tabular display.



```
X           Y
----        ----
0           0
10          97.4999
20          190
30          277.5
40          360
50          437.5
60          509.999
70          577.499
80          639.999
90          697.499
100         749.998
110         797.498
120         839.997
130         877.496
140         909.995
150         937.492

READY
-
```

The program responds with the tabulated values. In this equation, X represents time (seconds) and Y is velocity (centimeters per second).



```
X value      Y min = 150                    Y max = 900

0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150

READY
-
```

The program is run again, this time selecting graphical output. The Y range (here, velocity) is selected by the user, and can be either larger or smaller than the calculated range of values. For this run, four points lie outside the 150–900 range selected, so they are shown by closed circles at the edges of the plot.

PROGRAM LISTING

```
100 REM <DIFEQ>
110 REM First order differential equation
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 CLEAR 500: L$ = CHR$(182): T$ = CHR$(183)
200 PRINT CHR$(12), SPC(6); "DIFFERENTIAL ";
210 PRINT "EQUATION SOLVER": PRINT: PRINT: PRINT
220 PRINT TAB(8);: FOR J = 1 TO 51
230 PRINT CHR$(186);: NEXT J: PRINT: GOSUB 2050
240 S$ = "": GOSUB 2000
250 S$ = "The differential equation must be  "
260 GOSUB 2000
270 S$ = "defined at line 3000.  The form is:"
280 GOSUB 2000: GOSUB 2050
290 S$ = "3000 D = (your function of X and Y)"
300 GOSUB 2000: GOSUB 2050
310 S$ = "where D is equated to dy/dx.": GOSUB2000
320 GOSUB 2050: S$ = "": FOR J = 1 TO 46 STEP 2
330 S$ = S$ + " " + CHR$(132): NEXT J: GOSUB 2000
340 GOSUB 2050: GOSUB 2050
350 S$ = "If this has already been done, enter"
360 GOSUB 2000: S$ = "'C' to continue.": GOSUB2000
370 GOSUB 2050: S$="If not, enter 'S' to stop"
380 GOSUB 2000: S$="the program.  Then enter line"
390 GOSUB 2000: S$="3000 and restart the routine."
400 GOSUB 2000: GOSUB 2050: PRINT TAB(8);
410 FOR J = 1 TO 51: PRINT CHR$(179);: NEXT J
420 PRINT: PRINT TAB(8);: INPUT "Your response"; R$
430 IF R$ = "S" OR R$ = "s" THEN END
440 IF R$ <> "C" AND R$ <> "c" THEN 420
500 PRINT CHR$(12); "DIFFERENTIAL EQUATION SOLVER"
510 PRINT: PRINT: INPUT "Initial value of X"; XX
520 PRINT: INPUT "Initial value of Y"; YY
530 X = XX: Y = YY: GOSUB 3000
540 PRINT: INPUT "Step size for X"; DX
550 PRINT: INPUT "Final value of X"; XF
560 PRINT: PRINT "Do you want a graph ";
570 INPUT "(enter 'G') or table (enter 'T')"; F$
580 F$ = LEFT$(F$,1)
590 IF F$ = "T" OR F$ = "t" THEN F$ = "T": GOTO 750
600 IF F$ = "G" OR F$ = "g" THEN F$ = "G": GOTO 620
610 PRINT: PRINT "Say again?": GOTO 560
```

```
620 PRINT: INPUT "Minimum Y for graph scale"; YL
630 PRINT: INPUT "Maximum Y for graph scale"; YU
640 IF YU > YL THEN 700
650 PRINT: PRINT "How's that again?": GOTO 620
700 PRINT CHR$(12): PRINT "X value"; TAB(14);
710 PRINT "Y min ="; YL; TAB(48); "Y max ="; YU
720 PRINT TAB(10); L$;: FOR J = 10 TO 60
730 C = 187: IF J/5 = INT(J/5) THEN C = 173
740 PRINT CHR$(C);: NEXT J: PRINT T$: GOTO 760
750 PRINT CHR$(12); "X", "Y": PRINT "----", "----"
760 LC = 1: GOSUB 1500
800 Q = XX + DX: IF Q > XF + 1E-5 THEN PRINT: END
810 X = XX: Y = YY: GOSUB 3000
820 K0 = D: X = XX + DX/2: Y = YY + K0 * DX/2
830 GOSUB 3000: K1 = D:: Y = YY + K1 * DX/2
840 GOSUB 3000: K2 = D: X = XX + DX
850 Y = YY + K2 * DX: GOSUB 3000: K3 = D
860 DY = DX * (K0 + 2*K1 + 2*K2 + K3) / 6
870 YY = YY + DY: XX = XX + DX: GOSUB 1500
880 IF LC < 20 THEN LC = LC + 1: GOTO 800
900 PRINT: PRINT
910 INPUT "Hit 'RETURN' to continue"; R$
920 IF F$ = "G" THEN 700
930 GOTO 750
1500 IF F$ = "T" THEN PRINT XX, YY: RETURN
1510 F = (YY - YL)/(YU - YL)
1520 C = 136: V = 10 + 50 * F
1530 IF V < 10 THEN V = 10: C = 132
1540 IF V > 60 THEN V = 60: C = 132
1550 V = INT(V + 1.5): PRINT XX; TAB(10); L$;
1560 PRINT TAB(V); CHR$(C); TAB(62); T$: RETURN
2000 W = (INT(46 - LEN(S$))/2): PRINT TAB(8); L$;
2010 PRINT SPC(W); S$; TAB(58); T$: RETURN
2050 S$ = "": GOSUB 2000: RETURN
3000 PRINT: PRINT "Where's the equation?": STOP
3999 RETURN
```

EASY CHANGES

1. If you have already entered the differential equation and wish to skip the introductory output, add this line:

   175 GOTO 500

   This will immediately begin the input dialog.

2. If you wish to use negative stepsizes, line 800 must be changed to:

```
800 Q=XX+DX:IFQ<XF−1E−5 THEN PRINT:END
```

MAIN ROUTINES

150	String constant initialization.
200-440	Displays initial messages.
500-650	Gets user's inputs.
700-760	Initializes output display.
800-880	Computes each step.
900-930	Starts and stops output.
1500-1560	Plots graphical output or prints table line.
2000-2050	Subroutine to center output strings.
3000-3999	User-supplied routine to define D.

MAIN VARIABLES

L$,R$,S$	Output strings.
D	Value of dY/dX.
X,Y	Values of X,Y on current step.
XX,YY	Values of X,Y on last step.
DX	Stepsize in X.
XF	Final value of X.
F$	Output flag string (T=table, G=graph).
YL,YU	Minimum, maximum value of Y plot axis.
V	Tab position for graphical output.
C	CHR$ argument for graphical output.
K0,K1 K2,K3	Runge-Kutta coefficients.
R$	User response.
Q,F,W	Work variables.
J	Loop index.
LC	Line counter.

SUGGESTED PROJECTS

1. Modify the program to display the tabular output followed by the graphical output. During the tabular phase, the minimum and maximum values of Y can be saved and automatically used as the plot limits for the graphical output.

2. The value of dY/dX as a function of X is often a useful quantity to know. Modify the program to add it to the columnar display and/or the graphical display.
3. The inherent error in the calculation depends on the stepsize chosen. Most cases should be run with different stepsizes to insure that the errors are not large. If the answers do not change much, you can be reasonably certain that your solutions are accurate. Better yet, techniques exist to vary the stepsize during the calculation to insure that the error is sufficiently small during each step. Research these methods and incorporate them into the program.
4. The program can be easily broadened to solve a set of coupled, first order, differential equations simultaneously. This would greatly increase the types of problems that could be solved. Research this procedure and expand the program to handle it.

GRAPH

PURPOSE

Is a picture worth a thousand words? In the case of mathematical functions, the answer is often "yes." A picture, i.e. a graph, enables you to see the important behavior of a function quickly and accurately. Trends, minima, maxima, etc. become easy and convenient to determine.

GRAPH produces a 2-dimensional plot of a function that you supply. The function must be in the form Y = (any function of X). The independent variable X will be plotted along the abscissa (horizontal axis). The dependent variable Y will be plotted along the ordinate (vertical axis). You have complete control over the scaling that is used on the X and Y axes.

The program uses a special high-precision graphing technique which quadruples the normal resolution of the Sorcerer's graphics.

HOW TO USE IT

Before running the program, you must add one or more lines defining the function. This function, which must define Y in terms of X, will be a subroutine beginning at line 4000. The subroutine may be as simple or complex as desired, with one or several hundred lines. With X set to various values, the subroutine will be called to determine the corresponding Y value. Line 4999 is already a RETURN statement, so another needn't be added.

Having entered this subroutine, you are ready to run the program. It begins with a warning, reminding you that it assumes the function subroutine has already been entered at line 4000. The program then asks for the calculation range of X – that is, the maximum and minimum values of X that you desire to use. Values can be positive or negative, as long as the maximum is, in fact, larger than the minimum X value.

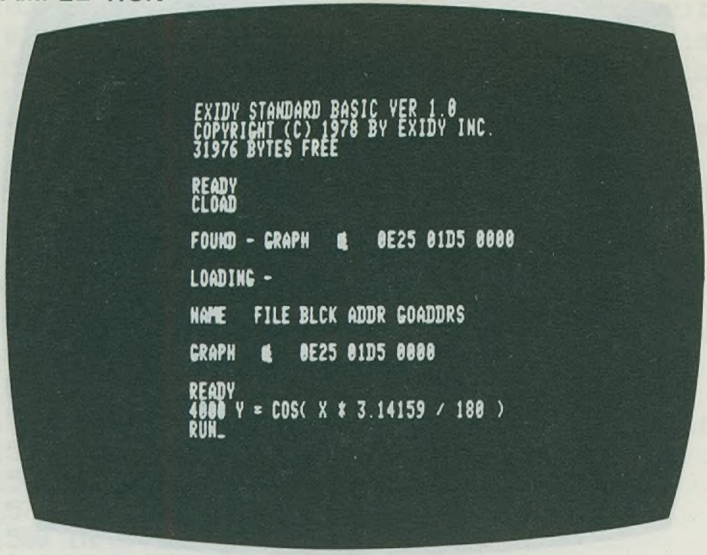
The plot range for Y is selected next. To help you, the program will calculate and display the maximum and minimum Y values that occur within the selected X range. You may choose any Y values for the plot range that you desire, either larger or smaller than the calculated Y range.

The program will now display the plot of your function over the selected X range. The Y-axis is 20 lines high, and the X-axis is 50 character spaces wide. The plot, though, is a field of points that is 40 "pixels" high and 100 wide, or four times as dense as normal. This is done with 16 specially-defined user characters which divide each character space into a 2-by-2 array of quarter-space-size pixels.

Axis values are displayed for the midrange and extremes, along with ten tick marks per axis. Y values which lie on the X- or Y-axis, or which are less than the selected Y minimum, are shown by one or more alternating blank spaces in the axis line. Y values that are greater than the selected Y maximum for the plot are shown just above the top line of the plot, using a special off-range character.

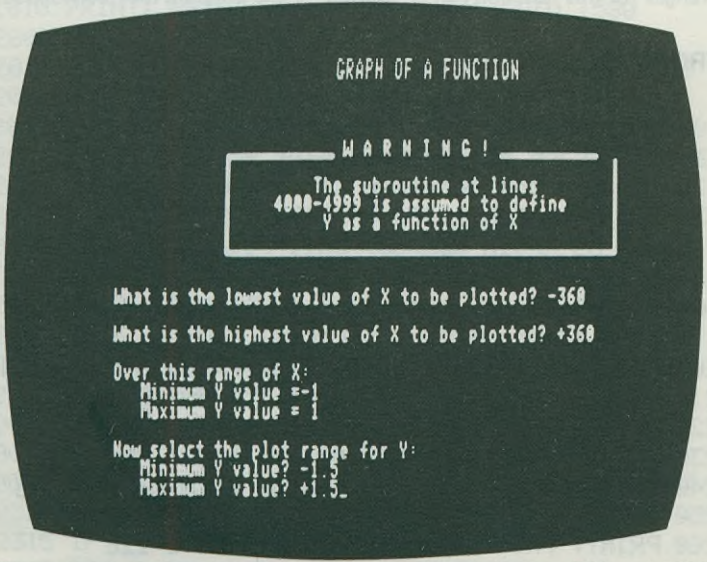
After the plot is drawn, the program will ask if you care to change the range of X or Y. Enter **Y** to restart, or **N** to end the program. The end will not cause the plot to scroll off-screen.

SAMPLE RUN



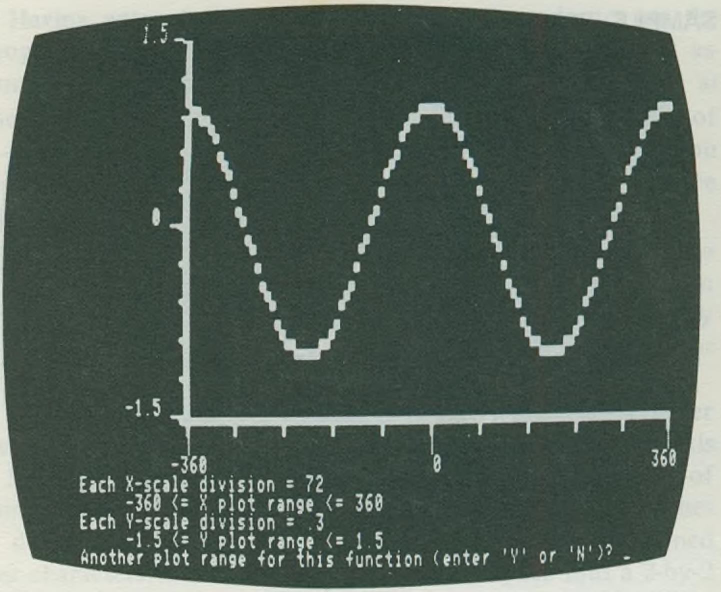
```
EXIDY STANDARD BASIC VER 1.0
COPYRIGHT (C) 1978 BY EXIDY INC.
31976 BYTES FREE

READY
CLOAD

FOUND - GRAPH   @   0E25 01D5 0000

LOADING -

NAME   FILE BLCK ADDR GOADDRS

GRAPH   @   0E25 01D5 0000

READY
4000 Y = COS( X * 3.14159 / 180 )
RUN_
```

After loading the program, the user enters the equation of the cosine curve at line 4000. **RUN** begins the program.



```
                    GRAPH OF A FUNCTION


              _______ W A R N I N G ! _______
             |                               |
             |     The subroutine at lines   |
             |  4000-4999 is assumed to define
             |      Y as a function of X     |
             |_______________________________|


What is the lowest value of X to be plotted? -360

What is the highest value of X to be plotted? +360

Over this range of X:
   Minimum Y value =-1
   Maximum Y value = 1

Now select the plot range for Y:
   Minimum Y value? -1.5
   Maximum Y value? +1.5_
```

The user requests a plot of the cosine function between −360 and +360 degrees. The Sorcerer calculates the values within this range, and displays the maximum and minimum. The Y range of the plot can be larger or smaller than the calculated range—here, a slightly larger range will be used.



```
Each X-scale division = 72
     -360 <= X plot range <= 360
Each Y-scale division = .3
     -1.5 <= Y plot range <= 1.5
Another plot range for this function (enter 'Y' or 'N')? _
```

The plot is displayed, along with relevant scaling information. Other ranges for this function can be plotted, or the program may be stopped.

PROGRAM LISTING

```
100 REM <GRAPH>
110 REM Graph of a function of X
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 CLEAR 500: DIM Z(3): GOSUB 5000
160 C = -2613
170 PB = 136
200 PRINT CHR$(12), SPC(9); "GRAPH OF A FUNCTION"
210 PRINT: PRINT: PRINT: L$ = "": GOSUB 1000
220 PRINT "What is the lowest value of X to be ";
230 INPUT "plotted"; XL: PRINT: PRINT "What is ";
240 INPUT "the highest value of X to be plotted";XU
250 IF XU > XL THEN 270
260 PRINT: PRINT "Bad X range!": GOTO 220
270 GOSUB 1500: GOSUB 2000: GOSUB 3000: GOSUB 3500
280 IF U$ <> "N" AND U$ <> "n" THEN 200
290 PRINT CHR$(23); CHR$(23);: END
1000 FOR J = 1 TO 11: L$ = L$ + CHR$(186): NEXT J
```

```
1010 L$ = L$ + " W A R N I N G ! ": FOR J = 1 TO 10
1020 L$ = L$ + CHR$(186): NEXT J: PRINT SPC(12); L$
1030 L$ = "            " + CHR$(182):R$=CHR$(183)
1040 B$ = L$: FOR J = 1 TO 38: B$ = B$ + " ":NEXT J
1050 B$ = B$ + R$: PRINT B$: PRINT L$; SPC(8);
1060 PRINT "The subroutine at lines"; SPC(7); R$
1070 PRINT L$; SPC(4); "4000-4999 is assumed to ";
1080 PRINT "define"; SPC(4); R$: PRINT L$; SPC(9);
1090 PRINT "Y as a function of X"; SPC(9); R$
1100 PRINT B$: PRINT SPC(12);: FOR J = 1 TO 40
1110 PRINT CHR$(179);: NEXT J: PRINT: PRINT: RETURN
1500 DX = (XU - XL)/100: X = XL: GOSUB 4000
1510 MN = Y: MX = Y: FOR J = 1 TO 100: X = XL+J*DX
1520 GOSUB 4000: IF Y > MX THEN MX = Y
1530 IF Y < MN THEN MN = Y
1540 NEXT J: PRINT: PRINT "Over this range of X:"
1550 PRINT "   Minimum Y value ="; MN
1560 PRINT "   Maximum Y value ="; MX: PRINT
1570 PRINT "Now select the plot range for Y:"
1580 INPUT "   Minimum Y value"; YL
1590 INPUT "   Maximum Y value"; YU
1600 IF YU > YL THEN RETURN
1610 PRINT: PRINT "Bad Y range!": GOTO 1580
2000 PRINT CHR$(12);: POKE C,210
2010 FOR J = 1 TO 19 STEP 2: POKE C - J*64, 209
2020 POKE C - (J+1)*64, 208: NEXT J
2030 FOR J = 1 TO 50: POKE C + J, 211: NEXT J
2040 FOR J=5 TO 50 STEP 5: PUKE C + J, 210: NEXT J
2050 XM = (XL + XU) / 2: YM = (YL + YU) / 2
2060 AV = YU: PRINT CHR$(17);: GOSUB 2310
2070 AV = YM: NL = 9: GOSUB 2300
2080 AV = YL: NL = 19: GOSUB 2300
2100 PRINT CHR$(17);: FOR J = 1 TO 22: PRINT ""
2110 NEXT J: A$ = CHR$(135): PRINT SPC(11); A$;
2120 PRINT SPC(24); A$; SPC(24); H$
2125 W = 12: AV = XL: GOSUB 2400
2130 W = 37: AV = XM: GOSUB 2400
2140 W = 62: AV = XU: GOSUB 2400: RETURN
2300 PRINT CHR$(17): FOR J=1 TO NL: PRINT: NEXT J
2310 A$ = STR$(AV): A = LEN(A$)
2315 IF A < 9 THEN PRINT SPC(9-A)
2320 PRINT A$; " "; CHR$(176): RETURN
2400 A$ = STR$(AV): A = LEN(A$): W = W - A/2
2410 IF W + LEN(A$) > 63 THEN W = 63 - A
2420 PRINT TAB(W); A$;: RETURN
```

```
3000 RF = 0: DY = (YU - YL) / 40
3010 FOR L = 0 TO 100: X = XL + L * DX: GOSUB 4000
3020 PY = 17 + INT((Y - YL)/DY + 0.5): PX = 23 + L
3030 IF PY < 17 THEN PY = 17: RF = 1
3040 IF PY > 58 THEN PY = 58: RF = 1
3050 GOSUB 6000: NEXT L: RETURN
3500 PRINT: PRINT "Each X-scale division ="; 10*DX
3510 PRINT TAB(5); XL; "<= X plot range <="; XU
3520 PRINT "Each Y-scale division ="; 4 * DY
3530 PRINT TAB(5); YL; "<= Y plot range <="; YU;
3540 IF RF = 0 THEN 3560
3550 PRINT "   RESTRICTED RANGE";
3560 PRINT: U$="": PRINT "Another plot range for ";
3570 INPUT "this function (enter 'Y' or 'N')"; U$
3580 RETURN
4000 Y = 0: REM -- INSERT USER FUNCTION HERE!!!!
4999 RETURN
5000 Z(0) = -1: FOR J = -512 TO -392 STEP 8: K = 0
5010 Z(K) = Z(K) + 1: IF Z(K) < 2 THEN 5030
5020 Z(K) = 0: K = K + 1: GOTO 5010
5030 W = 240 * Z(0) + 15 * Z(1)
5040 FOR K = J TO J + 3: POKE K, W: NEXT K
5050 W = 240 * Z(2) + 15 * Z(3)
5060 FOR K = J + 4 TO J + 7: POKE K, W: NEXT K
5070 NEXT J: POKE -384,255: FOR J = -383 TO -370
5080 POKE J,15: NEXT J: FOR J = -368 TO -353
5090 POKE J,255: NEXT J: FOR J = -365 TO -361
5100 POKE J,3: NEXT J: FOR J = -357 TO -353
5110 POKE J,0: NEXT J: POKE -369,255: RETURN
6000 PCOL = INT(PX/2): PROW = 29 - INT(PY/2)
6010 PIX = 0: IF PCOL <> PX/2 THEN PIX = 1
6020 IF PROW = 29 - PY/2 THEN PIX = PIX + 2
6030 PADD = 64 * PROW + PCOL - 3968
6040 PS = PEEK(PADD) - 192: IF PS < 0 THEN PS = 0
6045 IF PS > 15 THEN PS = 0
6050 FOR J = 3 TO 0 STEP -1: Z(J) = INT(PS/(2↑J))
6060 IF Z(J) > 0 THEN PS = PS - 2↑J
6070 NEXT J: PS = 192: Z(PIX) = 1
6080 IF PY > 17 THEN 6100
6090 PS = 195 - Z(PIX)*2↑PIX: GOTO 6150
6100 IF PY < 58 THEN 6120
6110 PS = PB: GOTO 6150
6120 IF PX > 23 THEN 6140
```

```
6130 PS = 202 - Z(PIX)*2↑PIX: GOTO 6150
6140 FOR J = 0 TO 3: PS = PS + Z(J) * 2↑J: NEXT J
6150 POKE PADD, PS: RETURN
```

EASY CHANGES

1. You may want the program to self-scale the Y axis for you. That is, you want it to use the minimum and maximum Y values that it finds as the limits on the Y axis. This can be accomplished by adding the following line:

   1565 YU=MX:YL=MN:RETURN

2. Do you sometimes forget to enter the subroutine at line 4000 despite the introductory warning? As is, the program will plot the straight line Y=0 if you do this. If you want a more drastic reaction to prevent this, change line 4000 to read

   4000 Y = 1/0

   Now, if you don't enter the actual subroutine desired, the program will stop and print the following message after you enter the X scaling values:

   ?/0 ERROR IN 4000

3. Would you like something different to be displayed as the off-range character? This can be done by changing the value of PB in line 170. The following table shows the correspondence between PB and the character used:

PB	*CHARACTER*
132	●
142	X
163	♦
177	■

For example, to have a diamond drawn as the off-range character, change line 170 to:

170 PB=163

MAIN ROUTINES

150-170	Initializes constants and special characters.
200-210	Displays introductory warning.

220-290	Mainline routine—gets X scaling from user and calls various subroutines.
1000-1110	Subroutines to display the introductory warning.
1500-1610	Subroutine which determines the minimum, maximum Y values; gets Y scale from user.
2000-2420	Subroutines to draw graph axes and scale labeling.
3000-3050	Subroutine to determine the plotting position for Y.
3500-3580	Subroutine which displays the scaling parameters.
4000-4999	User-supplied subroutine to evaluate Y as a function of X.
5000-5110	Subroutine to set up special plot and axis characters.
6000-6150	Subroutine to plot the function.

MAIN VARIABLES

XL,XM,XU	Lower, middle, and upper values of the X range.
YL,YM,YU	Lower, middle, and upper values of Y for the plot.
MN,MX	Calculated values of minimum and maximum Y in the selected range of X.
DX,DY	Scale increments of X and Y.
X,Y	Current values of X and Y.
PX,PY	Values of X and Y in plot scale units.
C	POKE address of plot origin (lower left corner).
PB	ASCII value of off-range character.
W	Work variable.
L$,R$,B$	Strings for warning box outline.
AV	Value for axis label.
A$	String representation of AV.
A	Length of A$.
J,K,L	Loop counters.
RF	Restricted range flag (equals 1 if plotted Y range is smaller than the calculated Y range).
NL	Number of lines to print to position cursor.
U$	User reply string.
Z	Array to hold state of four pixels in a character space.
PCOL, PROW	Screen character and row positions for a plot character group of four pixels.
PADD	PEEK/POKE address corresponding to PCOL, PROW.

PIX	Pixel number (0-3) of one quarter of a character space.
PS	Screen PEEK/POKE value corresponding to the character with the selected arrangement of "off" and "on" pixels.

SUGGESTED PROJECTS

1. Determine and display the values of X at which the minimum and maximum values of Y occur.
2. After the graph is plotted, allow the user to obtain the exact value of Y for any given X.
3. Allow the user to select the plot width (X direction).
4. Add an option to input and plot individual X-Y points, either separately or onto the completed plot of the function.

INTEGRATE

PURPOSE

The need to evaluate integrals occurs frequently in scientific and mathematical work. This program will numerically integrate a function that you supply, using a technique known as Simpson's rule. It will continue to grind out successive approximations of the integral until you are satisfied with the accuracy of the solution.

Mathematical integration will probably be a familiar term to those who have studied some higher mathematics. It is a fundamental subject of first-year calculus. The integral of a function between the limits x=ℓ (lower limit) and x=u (upper limit) represents the area under its curve; i.e. the shaded area in Figure 1.

We may approximate the integral by first dividing up the area into rectangular strips or segments. We can get a good estimate of the total integral by summing the areas of these segments by using a parabolic fit across the top. For those who understand some mathematical theory, Simpson's rule may be expressed as

$$\int_{x\,=\,\ell}^{x\,=\,u} f(x)\,dx \approx \frac{\Delta}{3}\left\{ f(\ell) + f(u) \right.$$

$$\left. + 4 \sum_{j=1}^{n/2} f[\ell + \Delta(2j-1)] + 2 \sum_{j=1}^{(n-2)/2} f[\ell + 2\Delta j] \right\}$$

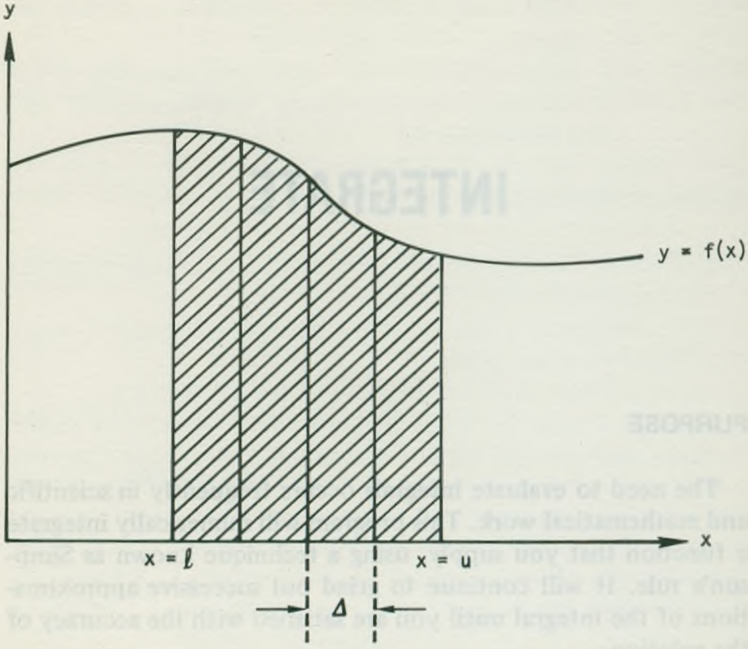


Figure 1. The Integral of f(x)

Here n is the number of segments into which the total interval is divided. Figure 1 uses n = 4.

For a good discussion of the numerical evaluation of integrals see the McCracken FORTRAN text listed in the bibliography. Don't let the word "FORTRAN" scare you away. The discussions in the book are independent of programming language with only some program examples written in FORTRAN. If you want to learn FORTRAN quickly, see McCabe's *QWIKTRAN*, also listed in the bibliography.

HOW TO USE IT

The program begins with a warning! This is to remind you that you should have already entered the subroutine to evaluate Y as a function of X. This subroutine must start at line 5000. More about it shortly.

You will then be asked to provide the lower and upper limits of the integration domain. Any numerical values are acceptable. It is not even necessary that the lower limit of X be smaller than the upper one.

The program will now begin displaying its numerical evaluations of the integral. The number of segments used in the calculation continually doubles. This causes the accuracy of the integral to increase at the expense of additional computation time. For most functions, you should see the value of the integral converging quickly to a constant (or near constant) value. This, of course, will be the best numerical evaluation of the integral at hand.

When you are satisfied with the accuracy of the solution, you must hit the **CTRL C** keys to terminate the program. If not, the program will run forever (assuming you can pay the electric bills). The amount of computation is approximately doubled each step. This means it will take the computer about the same amount of time to compute the next step that it took to compute *all* the previous steps. Thus, it will soon be taking the Sorcerer days to compute steps. Eventually, round-off errors begin degrading the results, causing a nice, constant, converged solution to change. However, you will probably lose patience before seeing it.

The function to be integrated can be as simple or as complicated as you desire. It may take one line or a few hundred lines of code. In any case, the subroutine to express it must start at line 5000. This subroutine will be continually called with the variable X set. When it returns, it should have set the variable Y to the corresponding value of the function for the given X. The subroutine must be able to evaluate the function at any value of X between the lower and upper bounds of the integration domain.

If your function consists of experimental data at discrete values of X, you must do something to enable the subroutine to evaluate the function at intermediate values of X. We recommend one of two approaches. First, you could write the subroutine to linearly interpolate the value of Y between the appropriate values of X. This will involve searching your data table for the pair of experimental X values that bound the value of X where the function is to be evaluated. Second, the program

CURVE, presented elsewhere in this section, can produce an approximate polynomial expression to fit your experimental data. This expression can then be easily entered as the subroutine beginning at line 5000.

By the way, Simpson's rule is *exact* for any polynomial of degree 3 or less. This means that if the function can be written in the form

Y = A∗X↑3 + B∗X↑2 + C∗X + D

where A, B, C, and D are constants, the program will calculate the integral exactly, even with only two segments.

SAMPLE RUN

The sample run illustrates the following integration:

$$\int_{x=0}^{x=1} \frac{4}{1+x^2}\, dx$$

This integral has the theoretical value of π (pi) as the correct answer. Pi, as you may know, has the value 3.1415926535 . . . Before the run is started, the above function is entered at line 5000.

```
5000 Y = 4 / (1 + X*X)
RUN


----------------------------------------
*                                      *
*       W A R N I N G !                *
*                                      *
*      The subroutine in lines         *
* 5000--5999 is assumed to define      *
*      Y as a function of X.           *
*                                      *
----------------------------------------


Lower limit of X? 0


Upper limit of X? 1
```

```
Number of     Integral
Segments      Value
---------     --------

2               3.13333
4               3.14157
8               3.14159
16              3.14159
32              3.14159

(CTRL C pressed)
```

PROGRAM LISTING

```
100 REM <INTEG>
110 REM Simpson's Rule integration
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
150 CLEAR 500: NSEG = 2
160 L$ = "          " + CHR$(182): R$=CHR$(183)
170 K$ = L$: FOR J = 1 TO 35: K$ = K$ + " ": NEXT J
180 K$ = K$ + R$: T$ = CHR$(186): B$ = CHR$(179)
200 GOSUB 2000
300 PRINT,: INPUT "Lower limit of X"; LX
310 PRINT: PRINT,;: INPUT "Upper limit of X"; UX
320 PRINT: PRINT: PRINT, "Number of", "Integral"
330 PRINT, "Segments", "Value"
340 PRINT, "---------", "--------": PRINT
400 SWIDTH = (UX - LX) / NSEG: T = 0
410 X = LX: GOSUB 5000: T = T + Y
420 X = UX: GOSUB 5000: T = T + Y
500 SUMS = NSEG / 2: TSUB = 0
510 FOR J = 1 TO SUMS
520 X = LX + SWIDTH*(2*J-1): GOSUB 5000
530 TSUB = TSUB + Y: NEXT J: T = T + 4*TSUB
600 SUMS = SUMS - 1: IF SUMS = 0 THEN 700
610 TSUB = 0: FOR J = 1 TO SUMS
620 X = LX + SWIDTH * 2 * J: GOSUB 5000
630 TSUB = TSUB + Y: NEXT J: T = T + 2*TSUB
700 VI = SWIDTH * T/3: PRINT, NSEG, VI
710 NSEG = 2 * NSEG
720 GOTO 400
2000 PRINT CHR$(12): PRINT SPC(17);
```

```
2010 PRINT "INTEGRATION BY SIMPSON'S RULE": PRINT
2020 PRINT
2030 PRINT SPC(13);: FOR J = 1 TO 37: PRINT T$;
2040 NEXT J: PRINT: PRINT K$: PRINT L$; SPC(9);
2050 PRINT "W A R N I N G !"; SPC(11); R$
2060 PRINT K$: PRINT L$; SPC(6);
2070 PRINT "The subroutine in lines"; SPC(6); R$
2080 PRINT L$; "  5000--5999 is assumed to define";
2090 PRINT "  "; R$: PRINT L$; SPC(7);
2100 PRINT "Y as a function of X."; SPC(7); R$
2110 PRINT K$: PRINT SPC(13);: FOR J = 1 TO 37
2120 PRINT B$;: NEXT J: PRINT: PRINT: PRINT
2130 RETURN
5000 Y = 1 / 0: REM -- Bombs if sub not entered!
5999 RETURN
```

EASY CHANGES

1. You might want the program to stop calculation after the integral has been evaluated for a given number of segments. Adding the following line will cause the program to stop after the integral is evaluated for up to 128 segments.

   715 IF NSEG > 128 THEN END

   Of course, you may use any value you wish instead of 128.
2. Perhaps you would like to see the number of segments change at a different rate during the course of the calculation. This can be done by modifying line 710. To increase the rate of change, try

   710 NSEG = 4*NSEG

   To change it at a constant (and slower) rate, try

   710 NSEG = NSEG + 20

   Be sure, however, that the value of NSEG is always even.

MAIN ROUTINES

150-180	Initializes constants and display strings.
200	Displays introductory messages and warning.
300-310	Gets integration limits from operator.
320-340	Displays column headings.
400-420	Computes integral contribution from end points.
500-530	Adds contribution from one summation.

600-630	Adds contribution from other summation.
700-720	Completes integral calculation and displays it. Increases number of segments and restarts calculation.
2000-2130	Warning display subroutine.
5000-5999	Operator-supplied subroutine to evaluate f(x).

MAIN VARIABLES

NSEG	Number of segments.
J	Loop index.
LX,UX	Lower, upper integration limit of x.
SWIDTH	Width of one segment.
T	Partial result of integral.
SUMS	Number of summations.
TSUB	Subtotal of summations.
VI	Value of integral.
X	Current value of x.
Y	Current value of the function y = f(x).
T$,B$,L$, R$,K$	String characters for warning box.

SUGGESTED PROJECTS

1. Research other techniques for numerical integration, such as the simpler trapezoid rule. Add a column of output, computing the integral with this new method as well. Compare the speed of the methods in converging to the correct answer.
2. Modify the program to compute answers to a higher precision, such as 12 digits. Hint: the trick behind higher-precision numbers is explained in the POWER program.

SIMEQN

PURPOSE

This program solves a set of simultaneous linear algebraic equations. It is capable of handling up to 15 equations in 15 unknowns. This type of problem often arises in scientific and numerical work. Algebra students encounter them regularly—many word problems can be solved by constructing the proper set of simultaneous equations.

The equations to be solved can be written mathematically as follows:

$$\begin{array}{ccccccccc}
A_{11}X_1 & + & A_{12}X_2 & + & \ldots & + & A_{1N}X_N & = & R_1 \\
A_{21}X_1 & + & A_{22}X_2 & + & \ldots & + & A_{2N}X_N & = & R_2 \\
\vdots & & \vdots & & & & \vdots & & \vdots \\
A_{N1}X_1 & + & A_{N2}X_2 & + & \ldots & + & A_{NN}X_N & = & R_N
\end{array}$$

N is the number of equations and, thus, the number of unknowns also. The unknowns are denoted X_1 through X_N.

Each equation also contains a coefficient multiplier for each unknown and a right-hand-side term. These coefficients (the A matrix) and the right-hand-sides (R_1 through R_N) must be constants—positive, negative, or zero. The A matrix is denoted with double subscripts. The first subscript is the equation number and the second one is the unknown that the coefficient multiplies.

HOW TO USE IT

The program will prompt you for all necessary inputs. First, it asks how many equations (and, thus, how many unknowns) comprise your set. This number must be at least 1 and no more than 15. Then, you must enter the coefficients and right-hand-sides for each equation. The program will request these, one at a time, continually indicating which term it is expecting next.

Once it has all your inputs, the program begins calculating the solution. This may take a little while if the number of equations is high. The program ends by displaying the answers. These, of course, are the values of each of the unknowns, X_1 through X_N.

If you are interested, the numerical technique used to solve the equations is known as Gaussian elimination. Row interchange to achieve pivotal condensation is employed. (This keeps maximum significance in the numbers.) Then, back substitution is used to arrive at the final results. This technique is much simpler than it sounds and is described well in the numerical analysis books referenced in the bibliography.

SAMPLE PROBLEM

Problem: A painter has a large supply of three different colors of paint: dark green, light green, and pure blue. The dark green is 30% blue pigment, 20% yellow pigment, and the rest base. The light green is 10% blue pigment, 35% yellow pigment, and the rest base. The pure blue is 90% blue pigment, no yellow pigment, and the rest base. The painter, however, needs a medium green to be composed of 25% blue pigment, 25% yellow pigment, and the rest base. In what percentages should he mix his three paints to achieve this mixture?

Solution: Let X_1 = percent of dark green to use,
X_2 = percent of light green to use,
X_3 = percent of pure blue to use.

The problem leads to these three simultaneous equations to solve:

$$\begin{aligned} 0.3\,X_1 + 0.1\,X_2 + 0.9\,X_3 &= 0.25 \\ 0.2\,X_1 + 0.35\,X_2 &= 0.25 \\ X_1 + X_2 + X_3 &= 1.0 \end{aligned}$$

The first equation expresses the amount of blue pigment in the mixture. The second equation is for the yellow pigment. The third equation states that the mixture is composed entirely of the three given paints. (Note that all percentages are expressed as numbers between zero and one.) The problem leads to the following use of SIMEQ.

SAMPLE RUN

```
          SIMULTANEOUS LINEAR EQUATION SOLVER

          How many unknowns in the set? 3

          The 3 unknowns will be denoted X1 thru X3

          ------------------------------

          Enter the parameters for equation # 1

          Coefficient of X1? .3
          Coefficient of X2? .1
          Coefficient of X3? .9
          Right hand side? .25

          ------------------------------

          Enter the parameters for equation # 2

          Coefficient of X1? .2
          Coefficient of X2? .35
          Coefficient of X3? Ø
          Right hand side? .25

          ------------------------------

          Enter the parameters for equation # 3

          Coefficient of X1? 1
          Coefficient of X2? 1
          Coefficient of X3? 1
          Right hand side? 1

          ------------------------------

The Sorcerer's solution is:

   X1 = .55
   X2 = .4
   X3 = .Ø5
READY
```

Thus, the painter should use a mixture of 55% dark green, 40% light green, and 5% pure blue.

PROGRAM LISTING

```
100 REM <SIMEQ>
110 REM Simultaneous linear equation
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
200 PRINT CHR$(12); "SIMULTANEOUS LINEAR ";
210 PRINT "EQUATION SOLVER": PRINT: PRINT
300 INPUT "How many equations in the set"; NEQ
310 IF NEQ >= 1 AND NEQ <= 15 THEN 330
320 PRINT "Sorry--between 1 and 15, please!"
325 PRINT: GOTO 300
330 DIM COEF(NEQ,NEQ), RSIDE(NEQ), SOL(NEQ)
340 PRINT: PRINT "The"; NEQ; "unknowns ";
350 PRINT "will be denoted X1 thru X";
360 PRINT MID$(STR$(NEQ),2,2)
370 FOR J = 1 TO NEQ: GOSUB 900
380 PRINT "Enter the parameters for equation #"; J
390 PRINT: FOR K = 1 TO NEQ
400 PRINT "Coefficient of X"; MID$(STR$(K),2,2);
410 INPUT COEF(J,K): NEXT K
420 INPUT "Right hand side"; RSIDE(J)
430 NEXT J: GOSUB 2000
500 GOSUB 900: PRINT "The Sorcerer's solution is:"
510 PRINT: FOR J = 1 TO NEQ: PRINT "  X";
520 PRINT MID$(STR$(J),2,2); " ="; SOL(J)
530 NEXT J: PRINT: END
900 PRINT: PRINT "--------------------------------"
910 PRINT: RETURN
2000 IF NEQ=1 THEN SOL(1)=RSIDE(1)/COEF(1,1):RETURN
2010 FOR K = 1 TO NEQ - 1: I = K + 1: L = K
2020 IF ABS(COEF(I,K)) > ABS(COEF(L,K)) THEN L = I
2030 IF I < NEQ THEN I = I + 1: GOTO 2020
2040 IF L = K THEN 2080
2050 FOR J = K TO NEQ: Q = COEF(K,J)
2060 COEF(K,J) = COEF(L,J): COEF(L,J) = Q: NEXT J
2070 Q = RSIDE(K): RSIDE(K) = RSIDE(L): RSIDE(L)= Q
2080 I = K + 1
2100 Q = COEF(I,K) / COEF(K,K): COEF(I,K) = 0
2110 FOR J = K+1 TO NEQ
2120 COEF(I,J) = COEF(I,J) - Q * COEF(K,J)
2130 NEXT J
2140 RSIDE(I) = RSIDE(I) - Q * RSIDE(K)
2150 IF i < NEQ THEN I = I + 1: GOTO 2100
```

```
2160 NEXT K
2200 SOL(NEQ) = RSIDE(NEQ) / COEF(NEQ,NEQ)
2210 FOR I = NEQ - 1 TO 1 STEP -1: Q = 0
2220 FOR J = I + 1 TO NEQ: Q = Q + COEF(I,J)*SOL(J)
2230 SOL(I) = (RSIDE(I) - Q) / COEF(I,I)
2240 NEXT J: NEXT I: RETURN
```

EASY CHANGES

You may sometimes be surprised to see the program fail completely and display this message:

```
?/0 ERROR IN 2200
```

This means that your input coefficients (the COEF array) were ill-conditioned and no solution was possible. This can arise from a variety of causes; e.g. if one equation is an exact multiple of another, or if *every* coefficient of one particular unknown is zero. If you would like the program to print a diagnostic message in these cases, add these lines:

```
2180 IF COEF(NEQ,NEQ) < > 0 THEN 2200
2190 PRINT "Bad data!":STOP
```

MAIN ROUTINES

200-210	Clears screen and displays program title.
300-430	Gets input from user and calculates the solution.
500-530	Displays the solution.
900-910	Subroutine to space and separate the output.
2000-2240	Subroutine to calculate the solution.
2000	Forms solution if NEQ=1.
2010-2160	Gaussian elimination. Interchanges rows to achieve pivotal condensation.
2200-2240	Back substitution.

MAIN VARIABLES

I,J,K,L	Loop indices and subscripts.
NEQ	Number of equations (thus, number of unknowns also).
COEF	Doubly-dimensioned array of the coefficients.
RSIDE	Array of right-hand-sides.

SOL	Array of the solution.
Q	Work variable.

SUGGESTED PROJECTS

1. The program modifies the COEF and RSIDE arrays while computing the answer. This means the original input cannot be displayed after it is input. Modify the program to save the information and enable the user to retrieve it after the solution is given.
2. Currently, a mistake in input cannot be corrected once the **RETURN** key is pressed after typing a wrong number. Modify the program to allow correction of previous input.
3. The restriction that NEQ not be greater than 15 is easily removed. Modify the program to allow NEQ>15 by changing lines 310 and 320.

STATS

PURPOSE

Ever think of yourself as a statistic? Many times we lament how we have become just numbers in various computer memories, or we simply moan at our insurance premiums. To most people, the word "statistics" carries a negative connotation. To invoke statistics is almost to be deceitful or, at least, dehumanizing. But really, we all use statistical ideas regularly. When we speak of things like "she was of average height" or the "hottest weather in years," we are making observations in statistical terms. It is difficult not to encounter statistics in our lives, and this book is no exception.

Of course, when used properly, statistics can be powerful analytical tools. STATS analyzes a set of numerical data that you provide. It will compile your list, order it sequentially, and determine several statistical parameters which describe it.

This should prove useful in a wide variety of applications. Teachers might determine grades by analyzing a set of test scores. A businessman might determine marketing strategy by studying a list of sales to clients. Little leaguers always like to pore over the current batting and pitching averages. You can probably think of many other applications.

HOW TO USE IT

Before receiving the data, the program will ask whether or not you wish to use identifiers with the data values. These identifiers can be anything associated with the data: e.g. names accompanying test scores, cities accompanying population values, corporations accompanying sales figures, etc. Enter **Y** or **N** to indicate yes or no regarding the use of identifiers.

Next, your data list must be entered. The program will prompt you for each value with a question mark. If identifiers are being used, you will be prompted for them before being asked for the associated data value. You may use any character strings you desire for identifiers. However, if you limit them to a maximum of 22 characters, the formatting of later output will be "cleaner."

Two special inputs, **END** and **BACK**, may be used at any time during this data input phase. They are applicable whether identifiers are being used or not. To signal the end of data, input the character string **END** in response to either question mark. You must, of course, enter at least one data value.

If you discover that you have made a mistake, the character string **BACK** can be used to back up the input process. This will cause the program to reprompt you for the previous entry. By successive uses of **BACK**, you can return to any previous position.

With the input completed, the program enters a command mode. You have five options as to how to continue the run:

1) List the data in the order input;
2) List the data in ranking order;
3) Display statistical parameters;
4) Start over;
5) End the program.

Simply input the number **1**, **2**, **3**, **4**, or **5** to indicate your choice. If one of the first three is selected, the program will perform the selected function and return to this command mode to allow another choice. This will continue until you choose to terminate or restart the run.

Options 1 and 2 provide lists of the data. Option 1 does it in the original input order while option 2 sorts the data from highest value to lowest. In either case the identifiers, if used, will be shown alongside their associated values.

The lists are shown 20 values at a time. This allows you to leisurely view data that might otherwise start scrolling off the screen. Simply hit **RETURN** for the next values. This starting and stopping can be repeated as often as desired. When the display is completed, you must again hit **RETURN** to reenter the command mode.

Option 3 produces a statistical analysis of your data. Various statistical parameters are calculated and displayed. The following is an explanation of some that may not be familiar to you.

Three measures of location, or central tendency, are provided. These are indicators of an "average" value. The *mean* is the sum of the values divided by the number of values. If the values are arranged in order from highest to lowest, the *median* is the middle value if the number of values is odd. If it is even, the median is the number halfway between the two middle values. The *midrange* is the number halfway between the largest and smallest values.

These measures of location give information about the average value of the data. However, they give no idea of how the data is dispersed or spread out around this "average." For that we need "measures of dispersion" or, as they are sometimes called, "measures of variation." The simplest of these is the *range* which is just the difference between the highest and lowest data values. Two other closely related measures of dispersion are given: the *variance* and the *standard deviation*. The variance is defined as:

$$VA = \frac{\sum_{i=1}^{N} (V_i - M)^2}{N - 1}$$

Here N is the number of values, V_i is value i, and M is the mean value. The standard deviation is simply the square root of the variance. We do not have space to detail a lengthy discussion of their theoretical use. For this, refer to the bibliography. Basically, however, the smaller the standard deviation, the more all the data tends to be clustered close to the mean value.

One word of warning—the first time option 2 or 3 is selected, the program must take some time to sort the data into numerical order. The amount of time that this requires depends upon how many items are on the list and how badly they are out of sequence. Average times are 20 seconds for 25 items, about 80 seconds for 50 items, and almost 6 minutes for 100 items. The Sorcerer will pause while this is occurring, so don't think it has hung up or fallen asleep! If you have several items on your list, this is the perfect chance to rob your refrigerator, make a quick phone call, or whatever.

SAMPLE RUN

```
STATISTICAL ANALYSIS ROUTINE

* * * * * * * * * * * * * * * *

This routine performs a statistical analysis on
a list of data values. It will order the list,
and calculate several statistical quantities
that describe the data.

The data values are entered one-by-one. At your
option, you may also enter an alphanumeric
identifier with each data value.

Do you wish to use identifiers (enter 'Y' or
'N')? Y

Data Input

* * * * * * * * * * * * * * * *

Data values must be entered now (up to 100
values).

For each data item, enter the ID word(s) first.
The associated data value entry then follows.

If you make an error, enter 'BACK' to re-enter
the last item.

When the list is complete, enter 'END' to
terminate the list.

Data item # 1
ID word(s)? Lou
Value? 87

Data item # 2
ID word(s)? Kevin
Value? 93

Data item # 3
ID word(s)? Sheri
Value? 84
```

```
Data item # 4
ID word(s)? Marie
Value? 86

Data item # 5
ID word(s)? Thomas
Value? 96

Data item # 6
ID word(s)? BACK

Data item # 5
ID word(s)? Thomas
Value? 97

Data item # 6
ID word(s)? END

Command Mode

* * * * * * * * * * * * * * * * *

Options menu:
   List data (input order) -- enter '1'
   List data (sorted) -- enter '2'
   Display analysis -- enter '3'
   Start over -- enter '4'
   End program -- enter '5'
Your choice? 2

Sorting . . .

Data List -- sorted order, 5 total entries

* * * * * * * * * * * * * * * * *

Count     Item Number     Value     ID Word(s)
-----     -----------     -----     ----------

1         5               97        Thomas
2         2               93        Kevin
3         1               87        Lou
4         4               86        Marie
5         3               84        Sheri

End of data.

Hit 'RETURN' for next screen?_
  :
(RETURN hit, Command Mode options listed)
  :
Your choice? 3
```

```
Statistical Analysis

* * * * * * * * * * * * * * * * *

Number of values--
   Positive     5
   Zero         0
   Negative     0
   Total        5

Data range--
   Minimum      84
   Maximum      97
   Range        13

Statistical quantities--
   Sum          447
   Mean         89.4
   Median       87
   Mid-range    90.5
   St. dev.     5.41293
   Variance     29.2998

Hit 'RETURN' for options menu?
   :
(RETURN hit, Command Mode options listed)
   :
Your choice? 5
```

PROGRAM LISTING

```
100 REM <STATS>
110 REM Statistical analysis
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
145 CLEAR 200
150 B1$ = "BACK": B2$ = "back": B3$ = "Back"
160 N$ = "": E1$ = "END": E2$ = "end": E3$ = "End"
170 MAXDV = 100: FLAG = -1: DOT$ = "": SFLAG = 0
180 DIM ID$(MAXDV), DV(MAXDV), Z(MAXDV)
190 FOR J = 1 TO 32
195 DOT$ = DOT$ + CHR$(132) + " ": NEXT J
200 PRINT CHR$(12), "STATISTICAL ANALYSIS ROUTINE"
210 PRINT: PRINT DOT$: PRINT: PRINT "This routine";
220 PRINT " performs a statistical analysis on a ";
230 PRINT "list of data": PRINT "values.  It will";
240 PRINT " order the list, and calculate several"
250 PRINT "statistical quantities that describe ";
```

```
260 PRINT "the data.": PRINT
270 PRINT "The data values are entered one-by-";
280 PRINT "one.  At your option, you may"
290 PRINT "also enter an alphanumeric identifier";
300 PRINT " with each data value.": PRINT
310 PRINT "Do you wish to use identifiers (enter";
320 INPUT " 'Y' or 'N')"; R$
330 IF R$ = "Y" OR R$ = "y" THEN FLAG = 1
340 IF R$ = "N" OR R$ = "n" THEN FLAG = 0
350 IF FLAG < 0 THEN PRINT "Huh?": PRINT: GOTO 310
400 PRINT CHR$(12); "Data Input": PRINT: PRINT DOT$
410 PRINT:PRINT "Data values must be entered now ";
420 PRINT "(up to"; MAXDV; "values).": PRINT
430 IF FLAG = 1 THEN 470
440 PRINT "Enter each value separately, in ";
450 PRINT "response to the question marks."
460 PRINT: GOTO 500
470 PRINT "For each data item, enter the ID word(";
480 PRINT "s) first.  The associated"
490 PRINT "data value entry then follows.": PRINT
500 PRINT "If you make an error, enter 'BACK' to ";
510 PRINT "re-enter the": PRINT "last item.":PRINT
520 PRINT "When the list is complete, enter 'END'";
530 PRINT " to terminate": PRINT "the list.":PRINT
540 PRINT DOT$: PRINT
550 IF ND < 1 THEN ND = 1
560 PRINT: PRINT "Data item #"; ND
570 IF FLAG = 0 THEN ID$(ND) = N$: GOTO 630
580 INPUT "ID word(s)"; R$
590 IF R$ = E1$ OR R$ = E2$ OR R$ = E3$ THEN 700
600 IF R$<>B1$ AND R$<>B2$ AND R$<>B3$ THEN 620
610 ND = ND - 1: GOTO 550
620 ID$(ND) = R$
630 INPUT "Value"; R$
640 IF R$ = E1$ OR R$ = E2$ OR R$ = E3$ THEN 700
650 IF R$<>B1$ AND R$<>B2$ AND R$<>B3$ THEN 670
660 ND = ND - 1: GOTO 550
670 DV(ND) = VAL(R$)
680 IF ND < MAXDV THEN ND = ND + 1: GOTO 550
690 PRINT "No more room!": ND = MAXDV + 1
700 ND = ND - 1: IF ND > 0 THEN 800
710 PRINT: PRINT "NO DATA -- run has been aborted."
720 PRINT: GOTO 810
800 PRINT CHR$(12);
810 PRINT "Command Mode": PRINT: PRINT DOT$: PRINT
```

```
820 PRINT "Options Menu:": PRINT
830 PRINT "   List data (input order) -- enter '1'"
840 PRINT "   List data (sorted) -- enter '2'"
850 PRINT "   Display analysis -- enter '3'"
855 PRINT "   Start over -- enter '4'"
860 PRINT "   End program -- enter '5'"
870 PRINT: INPUT "Your choice";R
880 IF R < 1 OR R > 5 THEN PRINT "Huh?": GOTO 870
890 IF R = 4 THEN 145
895 IF R = 5 THEN END
900 ON R GOSUB 950, 2000, 3000: GOTO 800
950 K = 1
960 IF R <> 2 THEN 1000
970 PRINT CHR$(12); "Data List -- sorted order,";
980 GOTO 1010
1000 PRINT CHR$(12); "Data List -- input order,";
1010 PRINT ND; "total entries"
1020 PRINT: PRINT DOT$: PRINT
1025 IF R = 2 THEN PRINT "Count",
1030 PRINT "Item Number", "Value",
1040 IF FLAG = 1 THEN PRINT "ID Word(s)";
1045 PRINT: PRINT "-----------", "-----------",
1050 IF R = 2 THEN PRINT "-----------",
1060 IF FLAG = 1 THEN PRINT "-----------";
1070 PRINT: PRINT: LC = 1
1080 L = K: IF R = 2 THEN L = Z(K)
1090 PRINT K,: IF R = 2 THEN PRINT L,
1095 PRINT DV(L), ID$(L): LC = LC + 1: K = K + 1
1100 IF LC < 21 AND K <= ND THEN 1080
1110 IF K <= ND THEN 1130
1120 PRINT: PRINT "End of data."
1130 PRINT: INPUT "Hit 'RETURN' for next screen";R$
1140 IF K > ND THEN RETURN
1150 GOTO 960
2000 IF SFLAG = 0 THEN GOSUB 2100
2010 GOSUB 950
2020 RETURN
2100 PRINT: PRINT "Sorting . . ."
2110 SFLAG = 1: FOR J = 1 TO ND: Z(J) = J: NEXT J
2120 IF ND = 1 THEN RETURN
2130 NM = ND - 1: FOR K = 1 TO ND: FOR J = 1 TO NM
2140 N1 = Z(J): N2 = Z(J+1)
2150 IF DV(N1) > DV(N2) THEN 2170
2160 Z(J+1) = N1: Z(J) = N2
2170 NEXT J: NEXT K
```

```
2180 RETURN
3000 IF SFLAG = 0 THEN GOSUB 2100
3005 PRINT CHR$(12); "Statistical Analysis": PRINT
3010 PRINT DOT$: PRINT: PRINT "Number of values--"
3020 NP = 0: NN = 0: NZ = 0: SS = 0: SV = 0
3030 FOR J = 1 TO ND: SV = SV + DV(J)
3040 SS = SS + DV(J) * DV(J)
3050 IF DV(J) < 0 THEN NN = NN + 1
3060 IF DV(J) = 0 THEN NZ = NZ + 1
3070 IF DV(J) > 0 THEN NP = NP + 1
3080 NEXT J: MV = SV / ND: VA = 0
3090 IF ND > 1 THEN VA = (SS - ND*MV*MV)/(ND - 1)
3100 SD = SQR(VA)
3110 PRINT "   Positive", NP
3120 PRINT "   Zero", NZ: PRINT "   Negative", NN
3125 PRINT "   Total", ND
3130 PRINT: PRINT "Data range--"
3140 PRINT "   Minimum", DV(Z(ND))
3150 PRINT "   Maximum", DV(Z(1))
3160 PRINT "   Range", DV(Z(1)) - DV(Z(ND))
3170 PRINT: PRINT "Statistical quantities--"
3180 PRINT "   Sum", SV: PRINT "    Mean", MV
3190 Q = INT(ND/2) + 1: MED = DV(Z(Q))
3200 IF ND/2 > INT(ND/2) THEN 3220
3210 MED = (DV(Z(Q)) + DV(Z(Q-1))) / 2
3220 PRINT "   Median", MED
3230 PRINT "   Mid-range", (DV(Z(1))+DV(Z(ND))) / 2
3240 PRINT "   St. dev.", SD
3250 PRINT "   Variance", VH: PRINT: PRINT
3260 INPUT "Hit 'RETURN' for options menu"; R$
3270 RETURN
```

EASY CHANGES

1. The program is dimensioned to allow a maximum of 100 data items. The total storage required depends upon the dimension parameter MAXDV in line 170. Increase the value to accommodate more than 100 values. If identifiers are used the value following CLEAR in line 145 may also need to be increased. This value controls the amount of memory reserved for strings.
2. Because of conflicts with identifiers on your list, you may wish to change the special strings that terminate or back up data entry. These are controlled in lines 150-160, using the

variables E1$, E2$, E3$, and B1$, B2$, B3$. If the string for backing up is to be **LAST**, for example, change line 150 to:

```
150 B1$ = “LAST”:B2$ = “Last”:B3$ = “last”
```

3. You may wish to sort the lists from smallest to largest, which is the opposite of the program as written. This can be done by changing the “greater than” sign (>) in line 2150 to a “less than” sign (<). This will, however, cause a few funny things to happen to the statistics. The real minimum value will be labeled as the maximum, and vice versa. Also, the range will have an erroneous minus sign. To cure these problems, make these additional changes:

```
3140 PRINT “  Minimum”, DV(Z(1))
3150 PRINT “  Maximum”, DV(Z(ND))
3160 PRINT “  Range”, DV(Z(ND))−DV(Z(1))
```

MAIN ROUTINES

145-195	Initializes constants and dimensioning.
200-350	Displays messages, determines if identifiers will be used.
400-690	Gets data from the user.
700-720	Checks that input contains at least one value.
800-900	Command mode–gets user’s next option and does a GOSUB or jump.
950-1150	Subroutine to list data.
2000-2020	Subroutine to list data in ranking order.
2100-2180	Subroutine to sort the list in ranking order.
3000-3270	Subroutine to calculate and display statistics.

MAIN VARIABLES

MAXDV	Maximum number of data values allowed.
ID$	String array of identifiers.
DV	Array of the data values.
Z	Array of the sorting order.
ND	Number of data values in current application.
FLAG	Flag on identifier usage (1=yes; 0=no).
B1$,B2$, B3$	Flag strings to back up the input.

E1$,E2$, E3$	Flag strings to signal end of the input.
N$	String for a null identifier.
R$	User input string.
NM	ND − 1.
R	Continuation option.
NP	Number of positive values.
NN	Number of negative values.
NZ	Number of zero values.
SV	Sum of the values.
SS	Sum of the squares of the values.
MV	Mean value.
MED	Median of the values.
VA	Variance.
SD	Standard deviation.
J,K	Loop indices, counters.
N1,N2	Possible data locations to be interchanged during sorting.
Q	Work variable.
DOT$	String of "dot" characters.
L	Item number.
SFLAG	Sort flag (1=sorted; 0=list not yet sorted).

SUGGESTED PROJECTS

1. The sorting algorithm used in the program is efficient only when the number of list items is fairly small—less than 25 or so. This is because it does not do checking along the way to see when the list becomes fully sorted. If your lists tend to be longer than 25 items, you might wish to use another sorting algorithm more appropriate for longer lists. Try researching other sorts and incorporating them into the program.
2. Because the INPUT statement is used when entering identifiers, commas cannot be used inside identifier names. BASIC will ignore anything entered beyond the comma. This can be circumvented if you use quotes around the identifier name, but it is easy to forget to do this. Modify the appropriate routine to remind the user.
3. Many other statistical parameters exist to describe this kind of data. Research them and add some that might be useful

to you. One such idea is classifying the data. This consists of dividing the range into a number of equal classes and then counting how many values fall into each class.

Section 6

Miscellaneous Programs

INTRODUCTION TO MISCELLANEOUS PROGRAMS

These programs show how simple programs can do interesting things. Most of them have a mathematical flavor. They are short and, as such, would be useful for study by those just learning BASIC or programming in general.

Monte Carlo simulation involves programming the computer to conduct an experiment. (It doesn't involve high-stakes gambling!). PI shows how this technique can be used to calculate an approximation to the famous mathematical constant pi.

PYTHA will find all right triangles with integral side lengths. A clever algorithm is utilized to do this.

Have you ever looked around your classroom or club meeting and wondered if any two people present had the same birthdate? BDAY will show you what the surprising odds are.

Very high precision arithmetic can be done on the Sorcerer with the proper know-how. POWER will calculate the values of integers raised to various powers, not to the Sorcerer's "normal" 6-digit precision, but up to 250 full digits of precision. It will also calculate factorial values with up to 250 digits.

BIRTHDAY

PURPOSE

Suppose you are in a room full of people. What is the probability that two or more of these people have the same birthday? How many people have to be in the room before the probability becomes greater than 50 percent? We are talking only about the month and day of birth, not the year.

This is a fairly simple problem to solve, even without a computer. With a computer to help with the calculations, it becomes very easy. What makes the problem interesting is that the correct answer is nowhere near what most people immediately guess. Before reading further, what do you think? How many people have to be in a room before there is better than a 50-50 chance of birthday duplication? 50? 100? 200?

HOW TO USE IT

When you **RUN** the program, it starts by displaying headings over two columns of numbers that will be shown. The left column is the number of people in the room, starting with one. The right column is the probability of birthday duplication.

For one person, of course, the probability is zero, since there is no one else with a possible duplicate birthday. For two people, the probability is simply the decimal equivalent of 1/365 (note that we assume a 365-day year, and an equal likelihood that each person could have been born on any day of the year).

What is the probability of duplication when there are three people in the room? No, not just 2/365. It's actually

$$1-\left(\frac{364}{365} * \frac{363}{365}\right)$$

This is simply one minus the probability of *no* duplicate birthdays.

The probability for four people is

$$1-\left(\frac{364}{365} * \frac{363}{365} * \frac{362}{365}\right)$$

The calculation continues like this, adding a new term for each additional person in the room. You will find that the result (probability of duplication) exceeds 50% surprisingly fast.

The program continues with the calculation until there are 60 people in the room. You will have to halt the program long before that to see the point where the probability first exceeds 50%.

SAMPLE RUN

```
COINCIDENT BIRTHDAY PROBABILITIES

Number of           Probability of
Persons             Duplication


 1                  Ø
 2                  2.73973E-Ø3
 3                  8.2Ø41E-Ø3
 4                  .Ø163558
 5                  .Ø271355
 6                  .Ø4Ø4624
 7                  .Ø562356
 8                  .Ø743352
 9                  .Ø946238
 10                 .116948
 11                 .141141
 12                 .167Ø25

(CTRL C pressed to halt)
```

PROGRAM LISTING

```
100 REM <BDAY>
110 REM Coincident birthday probability problem
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
200 PRINT CHR$(12);
210 PRINT "COINCIDENT BIRTHDAY PROBABILITIES"
220 PRINT CHR$(10); "Number of";
230 PRINT TAB(15); "Probability of"
240 PRINT "Persons"; TAB(15); "Duplication"
300 Q = 1
310 FOR N = 1 TO 60
320 PRINT N; TAB(15); 1 - Q
330 Q = Q * (365 - N) / 365
340 NEXT N
```

EASY CHANGES

Change the constant value of 60 in line 310 to alter the range of the number of people in the calculation. For example, change it to 100 and watch how fast the probability approaches 1.

MAIN ROUTINES

200-240	Displays headings.
300	Initializes Q to 1.
310-340	Calculates probability of no duplication, then displays probability of duplication.

MAIN VARIABLES

N	Number of people in the room.
Q	Probability of no duplication of birthdays.

SUGGESTED PROJECTS

1. Modify the program to allow for leap years in the calculation, instead of assuming 365 days in the year.
2. Change the program to calculate the probability of three or more persons in the room having coincident birthdays.

PI

PURPOSE

The Greek letter π (pi) represents probably the most famous constant in mathematical history. It occurs regularly in many different areas of mathematics. It is best known as the constant appearing in several geometric relationships involving the circle. The circumference of a circle of radius r is $2\pi r$, while the area enclosed by the circle is πr^2.

Being a transcendental number, pi cannot be expressed exactly by any number of decimal digits. To nine significant digits, its value is 3.14159265. Over many centuries, man has devised many different methods to calculate pi.

This program uses a valuable, modern technique known as computer simulation. The name "simulation" is rather self-explanatory; the computer performs an experiment for us. This is often desirable for many different reasons. The experiment may be cheaper, less dangerous, or more accurate to run on a computer. It may even be impossible to do in "real life." Usually, however, the reason is that the speed of the computer allows the simulation to be performed many times faster than actually conducting the real experiment.

This program simulates the results of throwing darts at a specially-constructed dartboard. Consider Figure 1 which shows the peculiar square dartboard involved. The curved arc, outlining the shaded area, is that of a circle with its center in the lower left-hand corner. The sides of the square and, thus, the radius of the circle, are considered to have a length of one.

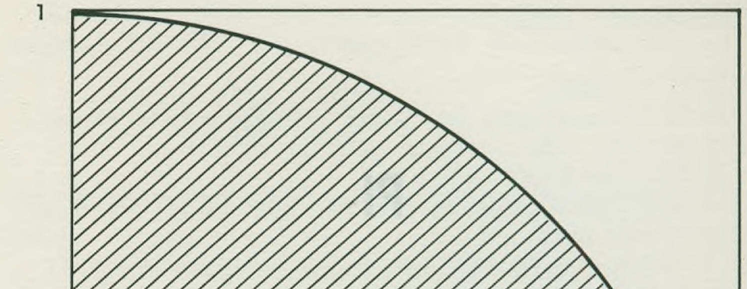


Figure 1. The PI Dartboard

Suppose we were able to throw darts at this square target in such a way that each dart had an equal chance of landing anywhere within the square. A certain percentage of darts would result in "hits," i.e. land in the shaded area. The expected value of this percentage is simply the area of the shaded part divided by the area of the entire square.

The area of the shaded part is one-fourth of the area that the entire circle would enclose if the arc were continued to completely form the circle. Recall that the area of a circle is πr^2 where r is the radius. In our case, r=1, and the area of the entire circle would simply be π. The shaded area of the dartboard is one-fourth of this entire circle and, thus, has an area of $\pi/4$. The area of the square is s^2, where s is the length of a side. On our dartboard, s=1, and the area of the whole dartboard is 1.

Now the expected ratio of "hits" to darts thrown can be expressed as:

$$\frac{\#\text{hits}}{\#\text{ thrown}} = \frac{\text{shaded area}}{\text{entire area}} = \frac{\pi/4}{1} = \frac{\pi}{4}$$

So we now have an experimental way to approximate the value of π. We perform the experiment and compute the ratio of "hits" observed. We then multiply this number by four and we have experimentally calculated π.

Instead of actually constructing the required dartboard and throwing real darts, we will let the Sorcerer do the job. The program "throws" each dart by selecting a separate random number between zero and one for the X and Y coordinates of each dart. This is accomplished by using the built-in RND function of BASIC. A "dart" is in the shaded area if $X^2 + Y^2 < 1$ for it.

The program grinds away, continually throwing darts and determining the ratio of "hits." This ratio is multiplied by four to arrive at an empirical approximation to π.

HOW TO USE IT

The program requires only one input from you. This is the "sample," i.e. how many darts it should throw before printing its current results. Any value of one or higher is acceptable.

After you input this number, the program will commence the simulation and display its results. A cumulative total of "hits" and darts thrown, as well as the current approximation to π, will be displayed for each multiple of the sample size.

This will continue until you press **CTRL C**. When you are satisfied with the total number of darts thrown, terminate the program execution.

SAMPLE RUN

```
DARTBOARD PI CALCULATOR

Sample interval? 150
```

```
DARTBOARD PI CALCULATOR

Hits__    Thrown_    π_____

118       150        3.14667
244       300        3.25333
361       450        3.20889
479       600        3.19333
588       750        3.136
694       900        3.08444
809       1050       3.0819
924       1200       3.08667

(CTRL C keys pressed to halt)
```

PROGRAM LISTING

```
100 REM <PI>
110 REM Dartboard pi calculation
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Phil Feldman & Tom Rugg
200 DARTS = 0: THITS = 0
300 PRINT CHR$(12);
310 PRINT "DARTBOARD PI CALCULATOR": PRINT
320 PRINT: INPUT "Sample interval"; SI
330 SI = INT(ABS(SI)): Q = RND(-SI)
340 PRINT CHR$(23); "Hits", "Thrown", CHR$(141)
350 FOR J = 1 TO 3: PRINT "------",: NEXT J
360 PRINT
400 GOSUB 500
410 THITS = THITS + HITS: DARTS = DARTS + SI
420 PI = 4 * THITS / DARTS
430 PRINT THITS, DARTS, PI
440 GOTO 400
500 HITS = 0: FOR J = 1 TO SI
510 X = RND(1): Y = RND(1)
520 IF (X*X + Y*Y) < 1 THEN HITS = HITS + 1
530 NEXT J: RETURN
```

EASY CHANGES

1. If you want the program to always use a fixed sample size, add line 315:

```
315 SI=150:Q=RND(-SI)
```

Of course, the value of 150 given here may be changed to whatever you wish. With this change, lines 320-330 are not needed and should be deleted.

2. If you want the program to stop by itself after a certain number of darts have been thrown, add the following two lines:

```
315 INPUT"TOTAL # DARTS TO THROW";ND
435 IF DARTS>=ND THEN END
```

This will ask the operator how many total darts should be thrown, and then terminate the program when they have been thrown.

MAIN ROUTINES

200	Initializes accumulators.
300-360	Gets operator input, displays column headings.
400-440	Calculates and displays results.
500-530	Throws SI darts and records number of "hits."

MAIN VARIABLES

DARTS	Total darts thrown.
THITS	Total "hits."
SI	Sample size for printing.
HITS	Number of hits in one group of SI darts.
PI	Calculated value of pi.
X,Y	Random-valued coordinates of a dart.
J	Loop index.
Q	Work variable.

SUGGESTED PROJECTS

1. Calculate the percentage error in the program's calculation of pi and display it with the other results.
2. The accuracy of this simulation is highly dependent on the quality of the random number generator. Try researching different algorithms for pseudo-random number generation. Then try incorporating them into the program. Change line 510 to use the new algorithm(s). This can actually be used as a test of the various random number generators. Gruenberger's

POWERS

PURPOSE

By now you have probably learned that the Sorcerer keeps track of six significant digits when dealing with numbers. For integers less than one million, the Sorcerer can retain the precise value of the number. For larger integers, the Sorcerer only keeps track of the most significant (leftmost) six digits, plus the exponent. This means, of course, that there is no way you can use the Sorcerer to deal with precise integers greater than one million, right?

Wrong.

This program calculates either factorials or successive powers of an integer, and can display precise results up to 250 digits long. By using a multiple-precision arithmetic technique, this program can tell you *exactly* what 973 to the 47th power is, for example.

HOW TO USE IT

The first input specifies the operator's choice of powers or factorials. The program next asks for the desired maximum number of digits, which may be any integer between 1 and 250. For example, entering **40** will yield answers up to 40 digits long.

If factorials are selected, the program will print 1! (one factorial), 2! (two factorial), and so on. If you've forgotten, 3! is 3*2*1, and 5! is 5*4*3*2*1, or 24.

If powers are desired, the program will ask for the base to be used. Output will consist of that base raised, successively, to the first, second, third, etc. powers.

Output continues until a value contains more than the requested maximum number of digits. At that point, the program will ask whether to terminate, or to start over.

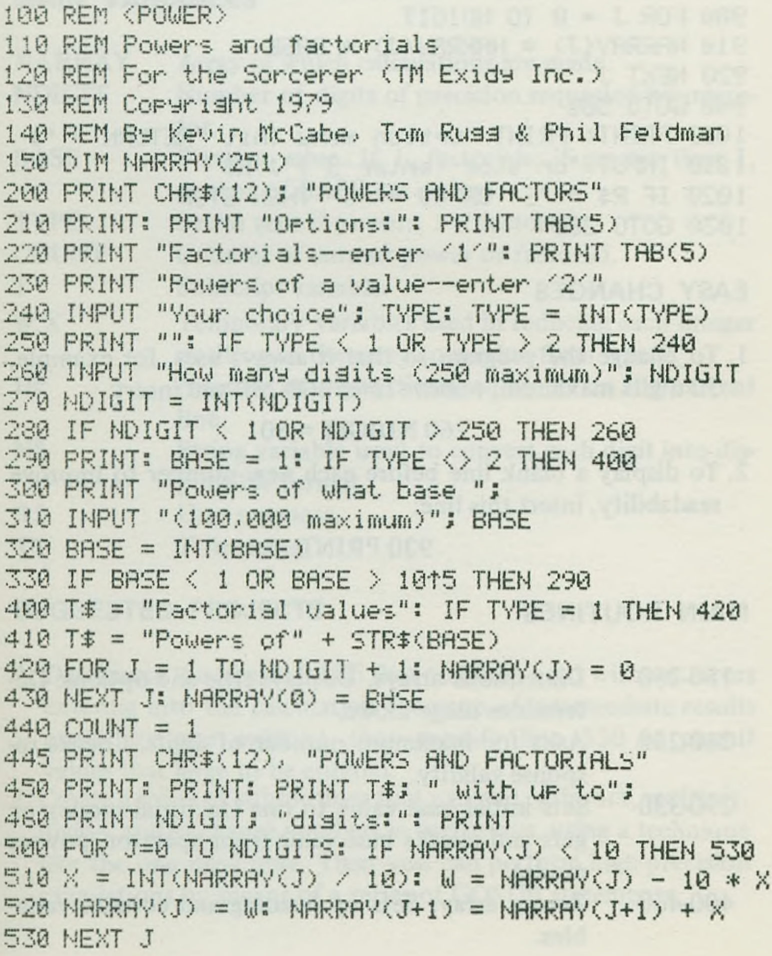
Note that, as the maximum number of digits requested is increased, the time required to calculate each value increases as well.

SAMPLE RUN

```
POWERS AND FACTORS

Options:
      Factorials--enter '1'
      Powers of a value--enter '2'
Your choice? 1

How many digits (250 maximum)? 15

Factorial values with up to 15 digits:

 1!            1
 2!            2
 3!            6
 4!            24
 5!            120
 6!            720
 7!            5040
 8!            40320
 9!            362880
 10!           3628800
 11!           39916800
 12!           479001600
 13!           6227020800
 14!           87178291200
 15!           1307674368000
 16!           20922789888000
 17!           355687428096000

Option menu (hit 'RETURN') or stop
(enter 'S')? _ (RETURN hit)

Options:
      Factorials--enter '1'
      Powers of a value--enter '2'
Your choice? 2

How many digits (250 maximum)? 18
```

```
Powers of what base (100,000 maximum)? 12345

Powers of 12345 with up to 18 digits:

12345 ↑ 1     12345
12345 ↑ 2     152399025
12345 ↑ 3     1881365963625
12345 ↑ 4     23225462820950625

Option menu (hit 'RETURN') or stop
(enter 'S')? S
```

PROGRAM LISTING



```
100 REM <POWER>
110 REM Powers and factorials
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
150 DIM NARRAY(251)
200 PRINT CHR$(12); "POWERS AND FACTORS"
210 PRINT: PRINT "Options:": PRINT TAB(5)
220 PRINT "Factorials--enter '1'": PRINT TAB(5)
230 PRINT "Powers of a value--enter '2'"
240 INPUT "Your choice"; TYPE: TYPE = INT(TYPE)
250 PRINT "": IF TYPE < 1 OR TYPE > 2 THEN 240
260 INPUT "How many digits (250 maximum)"; NDIGIT
270 NDIGIT = INT(NDIGIT)
280 IF NDIGIT < 1 OR NDIGIT > 250 THEN 260
290 PRINT: BASE = 1: IF TYPE <> 2 THEN 400
300 PRINT "Powers of what base ";
310 INPUT "(100,000 maximum)"; BASE
320 BASE = INT(BASE)
330 IF BASE < 1 OR BASE > 10↑5 THEN 290
400 T$ = "Factorial values": IF TYPE = 1 THEN 420
410 T$ = "Powers of" + STR$(BASE)
420 FOR J = 1 TO NDIGIT + 1: NARRAY(J) = 0
430 NEXT J: NARRAY(0) = BASE
440 COUNT = 1
445 PRINT CHR$(12), "POWERS AND FACTORIALS"
450 PRINT: PRINT: PRINT T$; " with up to";
460 PRINT NDIGIT; "digits:": PRINT
500 FOR J=0 TO NDIGITS: IF NARRAY(J) < 10 THEN 530
510 X = INT(NARRAY(J) / 10): W = NARRAY(J) - 10 * X
520 NARRAY(J) = W: NARRAY(J+1) = NARRAY(J+1) + X
530 NEXT J
```

```
600 J = NDIGIT + 1
610 IF NARRAY(J) = 0 THEN J = J - 1: GOTO 610
620 IF J >= NDIGIT THEN 1000
700 DP = 0: IF TYPE = 2 THEN PRINT BASE; "↑";
705 PRINT COUNT;: IF TYPE=1 THEN PRINT CHR$(8);"!";
710 PRINT TAB(15);
715 N$ = RIGHT$(STR$(NARRAY(J)), 1)
720 DP = DP + 1: IF DP <= 45 THEN 740
730 DP = 0: PRINT: PRINT TAB(15);
740 PRINT N$;
750 J = J - 1: IF J >= 0 THEN 710
800 IF TYPE = 1 THEN BASE = BASE + 1
810 COUNT = COUNT + 1: PRINT ""
900 FOR J = 0 TO NDIGIT
910 NARRAY(J) = NARRAY(J) * BASE
920 NEXT J
940 GOTO 500
1000 PRINT: PRINT "Option menu (hit ´RETURN´) ";
1010 INPUT "or stop (enter´S´)"; R$
1020 IF R$ = "S" OR R$ = "s" THEN STOP
1030 GOTO 200
```

EASY CHANGES

1. To change the program so that it always uses, for example, 50 digits maximum, remove lines 260-280 and insert:

   260 NDIGIT = 50

2. To display a blank line before each new number to improve readability, insert this line:

   930 PRINT

MAIN ROUTINES

150-250 Dimensions arrays. Displays title and options. Determines usage mode.

260-280 Asks for maximum number of digits. Checks response validity.

290-330 Sets initial base value to one (factorial mode), or gets and checks base value from operator (powers mode).

400-460 Zeroes array. Sets up heading and counter variables.

500-530	Performs "carrying" in NARRAY, so that no element exceeds 9.
600-620	Checks number of digits used. Scans NARRAY for most significant digit.
700-750	Converts NARRAY values to strings, removing leading blanks. Outputs up to 45 characters per line.
800-810	Updates counter. Updates factorial base.
900-940	Multiplies all values in NARRAY by base value. Returns to line 500.
1000-1030	Gets user's choice of continuation options.

MAIN VARIABLES

NARRAY	Array in which calculations are made.
NDIGIT	Number of digits of precision requested by operator.
BASE	Starting value. If 1, factorials; if greater than 1, powers of BASE are output.
TYPE	Set to zero if powers, 1 if factorial.
COUNT	Counter of current power or factorial.
J	Subscript variable.
W,X	Temporary variables used in reducing each integer position in the array to a value from 0 to 9.
DP	Number of digits displayed so far on the current line.
N$	String variable used to convert each digit into displayable format.
R$	User response.
T$	Title string.

SUGGESTED PROJECTS

1. Determine the largest BASE that could be used without errors entering into the calculation (because of intermediate results exceeding one million), then modify line 330 to permit values that large to be entered.
2. Create a series of subroutines that can add, subtract, multiply, divide, and exchange numbers in two arrays, using a technique like the one used here. Then you can perform high-precision calculations by means of a series of GOSUB statements.

PYTHAG

PURPOSE

Remember the Pythagorean Theorem? It says that the sum of the squares of the two legs of a right triangle is equal to the square of the hypotenuse. Expressed as a formula, it is $a^2 + b^2 = c^2$. The most commonly remembered example of this is the 3-4-5 right triangle ($3^2 + 4^2 = 5^2$). Of course, there are infinite other right triangles.

This program displays integer values of a, b, and c that result in right triangles.

HOW TO USE IT

To use this program, all you need to do is **RUN** it and watch the "Pythagorean triplets" (sets of values for a, b, and c) come out. The program displays 12 sets of values on each screen, and then waits for you to press **RETURN** before it continues with the next 12. It will go on indefinitely until you enter an **S**.

The left-hand column shows the count of the number of sets of triplets produced, and the other three columns are the values of a, b, and c.

The sequence in which the triplets are produced is not too obvious, so we will explain how the numbers are generated. It has been proved that the following technique will generate all *primitive* Pythagorean triplets. ("Primitive" means that no set is an exact multiple of another.) If you have two positive integers called R and S such that:

1. R is greater than S,
2. R and S are of opposite parity (one is odd and the other is even), and
3. R and S are relatively prime (they have no common integer divisors except 1),

then a, b, and c can be found as follows:

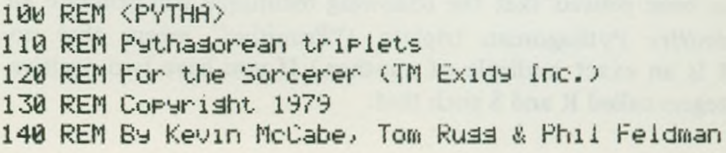
$a = R^2 - S^2$
$b = 2RS$
$c = R^2 + S^2$

The program starts with a value of 2 for R. It generates all possible S values for that R (starting at R−1 and then decreasing) and then adds one to R and continues. So, the first set of triplets is created when R is 2 and S is 1, the second set when R is 3 and S is 2, and so on.

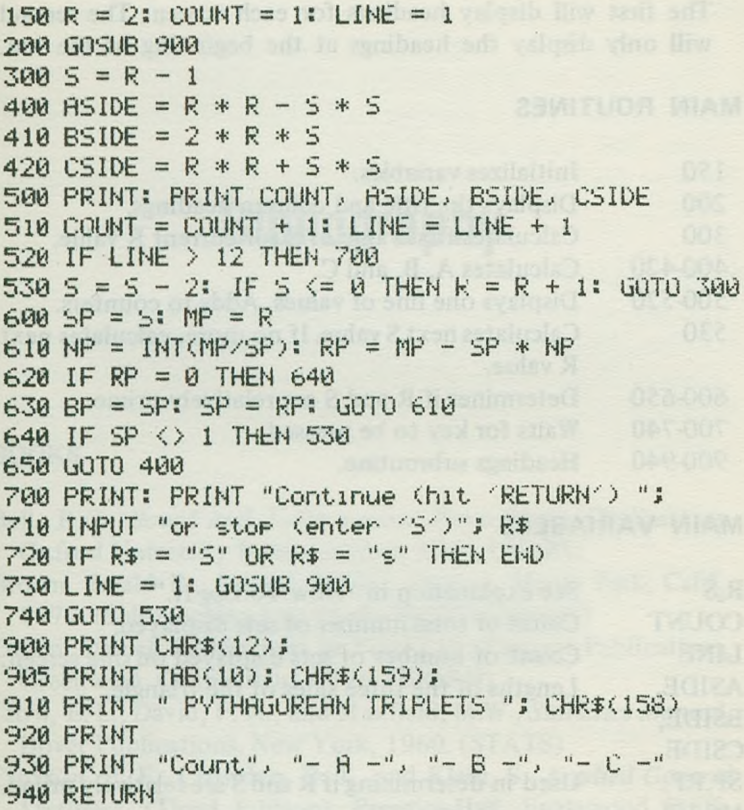
SAMPLE RUN

```
PYTHAGOREAN TRIPLETS

Count          - A -          - B -          - C -

1              3              4              5
2              5              12             13
3              7              24             25
4              15             8              17
5              9              40             41
6              21             20             29
7              11             60             61
8              35             12             37
9              13             84             85
10             33             56             65
11             45             28             53
12             15             112            113

Continue (hit 'RETURN') or stop (enter 'S')? S
```

PROGRAM LISTING



```
100 REM <PYTHA>
110 REM Pythagorean triplets
120 REM For the Sorcerer (TM Exidy Inc.)
130 REM Copyright 1979
140 REM By Kevin McCabe, Tom Rugg & Phil Feldman
```



```
150 R = 2: COUNT = 1: LINE = 1
200 GOSUB 900
300 S = R - 1
400 ASIDE = R * R - S * S
410 BSIDE = 2 * R * S
420 CSIDE = R * R + S * S
500 PRINT: PRINT COUNT, ASIDE, BSIDE, CSIDE
510 COUNT = COUNT + 1: LINE = LINE + 1
520 IF LINE > 12 THEN 700
530 S = S - 2: IF S <= 0 THEN R = R + 1: GOTO 300
600 SP = S: MP = R
610 NP = INT(MP/SP): RP = MP - SP * NP
620 IF RP = 0 THEN 640
630 BP = SP: SP = RP: GOTO 610
640 IF SP <> 1 THEN 530
650 GOTO 400
700 PRINT: PRINT "Continue (hit 'RETURN') ";
710 INPUT "or stop (enter 'S')"; R$
720 IF R$ = "S" OR R$ = "s" THEN END
730 LINE = 1: GOSUB 900
740 GOTO 530
900 PRINT CHR$(12);
905 PRINT TAB(10); CHR$(159);
910 PRINT " PYTHAGOREAN TRIPLETS "; CHR$(158)
920 PRINT
930 PRINT "Count", "- A -", "- B -", "- C -"
940 RETURN
```

EASY CHANGES

1. Alter the starting value of R in line 150. Instead of 2, try 50 or 100.
2. If you want, you can change the number of sets of triplets displayed on each screen, Change the 12 in line 520 to 10, for example. You probably won't want to try a value greater than 12, since that would cause the column headings to roll off the screen.
3. To make the program continue without requiring you to press a key for the next screen of values, insert either of these lines:

```
700 LINE=1:PRINT:GOSUB 905:GOTO 530
```

or

```
515 GOTO 530
```

The first will display headings for each screen. The second will only display the headings at the beginning of the run.

MAIN ROUTINES

150	Initializes variables.
200	Displays the title and column headings.
300	Calculates first value of S for current R value.
400-420	Calculates A, B, and C.
500-520	Displays one line of values. Adds to counters.
530	Calculates next S value. If no more, calculates next R value.
600-650	Determines if R and S are relatively prime.
700-740	Waits for key to be pressed.
900-940	Headings subroutine.

MAIN VARIABLES

R,S	See explanation in "How To Use It."
COUNT	Count of total number of sets displayed.
LINE	Count of number of sets displayed on one screen.
ASIDE, BSIDE, CSIDE	Lengths of the three sides of the triangle.
SP,RP, MP,NP	Used in determining if R and S are relatively prime.
R$	Key pressed by operator to continue.

SUGGESTED PROJECTS

1. In addition to displaying the counter and lengths on each line, display R and S.
2. Because this program uses integer values that get increasingly larger, eventually some will exceed the Sorcerer's integer capacity and produce incorrect results. Can you determine when this will be? Modify the program to stop when this occurs.

Bibliography

BOOKS

Bell, R.C., *Board and Table Games From Many Civilizations*, Oxford University Press, London, 1969, (WARI)

Brown, Jerald R., *Instant BASIC*, Dymax, Menlo Park, Calif., 1977. (Self-teaching text on the BASIC language)

Cohen, Daniel, *Biorhythms in Your Life*, Fawcett Publications, Greenwich, Connecticut, 1976. (BIORH)

Crow, E. L., David, F. A., and Maxfield, M.W., *Statistics Manual,* Dover Publications, New York, 1960. (STATS)

Croxton, F. E., Crowden, D. J., and Klein, S., *Applied General Statistics*, (Third Edition), Prentice-Hall, Englewood Cliffs, N.J., 1967. (STATS)

Gruenberger, Fred J., and Jaffray, George, *Problems for Computer Solution*, John Wiley and Sons, New York, 1965. (BDAY, PI)

Gruenberger, Fred J., and McCracken, Daniel D., *Introduction to Electronic Computers*, John Wiley and Sons, New York, 1961. (MILES, PI, PYTHA, WARI as Oware)

Hildebrand, F. B., *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956. (CURVE, DIFEQ, INTEG, SIMEQ)

Kuo, S. S., *Computer Applications to Numerical Methods*, Addison-Wesley, Reading Massachusetts, 1972. (CURVE, DIFEQ, INTEG, SIMEQ)

McCabe, C. Kevin, *QWIKTRAN: Quick FORTRAN for Micros, Minis, and Mainframes*, dilithium Press, Portland, Oregon, 1979. (Self-teaching text on the FORTRAN language)

McCracken, Daniel D., and Dorn, W. S., *Numerical Methods And FORTRAN Programming*, John Wiley and Sons, New York, 1964. (CURVE, DIFEQ, INTEG, SIMEQ)

Shefter, Harry, *Faster Reading Self-Taught*, Washington Square Press, New York, 1960. (TACHI)

Sorcerer Technical Manual, Exidy Inc., Sunnyvale, California, 1979.

PERIODICALS

Feldman, Phil, and Rugg, Tom, "Pass the Buck," *Kilobaud*, July 1977, pp. 90-96. (PICKS)

McCabe, C. Kevin, "Conjure up a GET Command for Sorcerer," *Kilobaud*, April 1980, pp. 46-47. (MATH, WALLS, RACER)

All of the programs in this book have been tested carefully and are working correctly to the best of our knowledge. However, we take no responsibility for any losses which may be suffered as a result of errors or misuse. You must bear the responsibility of verifying each program's accuracy and applicability for your purposes.

If you want to get a copy of an errata sheet that lists corrections for any errors or ambiguities we have found to date, send one dollar ($1.00) and a self-addressed stamped envelope (SASE) to the address below. Ask for errata for this book (by name). We hope we won't have any errors to tell you about, in which case we'll try to send you some other worthwhile information about the Sorcerer.

If you think you've found an error, please let us know. If you want an answer, include a SASE.

C. Kevin McCabe, Tom Rugg,
and Phil Feldman
Errata—32 Programs for the Sorcerer
c/o dilithium Press
P.O. Box 606
Beaverton, Oregon 97075

About the Book. . . ,

Chock full of programs with practical applications, educational uses, games and graphics, this book is a *must* for the Exidy Sorcerer user. Each of the 32 chapters fully documents a different program, and you can adapt the programs by making changes the authors suggest.

About the Authors. . .

TOM RUGG and PHIL FELDMAN have a long-standing association with the microcomputer field. Together they wrote the popular column "Games and Things" in SOUTHERN CALIFORNIA COMPUTER SOCIETY INTERFACE, and have written numerous articles for other publications. They are authors of several books published by dilithium Press.

KEVIN McCABE received his undergraduate degree in aeronautical and astronautical engineering and then proceeded to become involved in everything from windmill design to the NASA Space Shuttle. He now works for a Chicago law firm as a specialist in aviation accidents. This is his second book for dilithium Press.

0-918398-35-5

dilithium Press
P.O. Box 606
Beaverton, OR 97075